



UNIVERSITÀ DI PISA

Open source tools for usability analysis of medical devices

Prof. Cinzia Bernardeschi

*Department of Information Engineering
University of Pisa*

cinzia.bernardeschi@unipi.it

Outline

1. Use errors in medical devices
2. PVSio-web framework
 1. The Prototype builder
 2. The Simulator
 3. How to build a prototype: a simple example
 4. Available demos: Bbraun infusion pump, Alaris GP, Radical7 monitoring system, Stellant contrast media injector
3. A case study on infusion pumps
4. Building a prototype of a contrast media injector
5. Prototypes of more complex systems
 - ICE
 - Heart-pacemaker
6. Conclusions

1. Use errors in medical devices

Medical devices

Most medical devices used in hospital and home care are interactive, they are controlled by software that governs key aspects of the user interface and performs key safety functions



Infusion pumps

deliver fluids, such as nutrients and medications, into a patient's body in controlled amounts



Medical monitors

monitor vital parameters, blood pressure, pulse oximetry and respiratory rate



Contrast media injectors

inject contrast and saline into the bloodstream of patients (used, for example, in CT)

Use errors in medical devices

Use errors with infusion devices, as well as with other medical devices, is a known source of incidents in healthcare

- Design flaws can induce use errors in the data entry system of a device (e.g., silently discarding decimal point key presses for certain range of values)
- Critical control-flows of functions and different operating modalities can induce use errors in the programming of a device (e.g., interleaving of operations of different operating modes)

Main points:

- (i) understanding of the design challenges with user interface software for medical systems
- (ii) tools and techniques for design and analysis of software incorporated in interactive medical systems

Usability of medical devices

Human factors play a fundamental role in reducing use errors in medical devices: the user must be considered in the device design and usability tests are fundamental for limiting the use errors.

User interface issues can potentially lead to use errors, e.g., complicated menu structures have a huge potential for accidentally making an error



if medical devices are to be used safely, it is important that “user interface software” is designed to make the device easy to use and mistakes made by users are corrected

Two reference standards for usability of medical devices

ANSI/AAMI HE75:2009, Human factors engineering – Design of medical devices,

The Association for the Advancement of Medical Instrumentation, American National Standards Institute Inc.

“This recommended practice covers general human factors engineering (HFE) principles, specific HFE principles geared towards certain user-interface attributes, and special applications of HFE (e.g., connectors, controls, visual displays, automation, software–user interfaces, hand tools, workstations, mobile medical devices, home health care devices).”

IEC 62366-1:2015 INTERNATIONAL STANDARD, Medical devices – Part 1: Application of usability engineering to medical devices, The International Electrotechnical Commission.

“This part of IEC 62366 specifies a process for a manufacturer to analyse, specify, develop and evaluate the **USABILITY** of a **MEDICAL DEVICE** as it relates to **SAFETY**. This usability engineering (**HUMAN FACTORS ENGINEERING**) process permits the manufacturer to assess and mitigate **RISKS** associated with correct use and use errors, i.e., normal use. “

Software complexity in medical devices

Due to more sophisticated software, recalls have increased since 2006

Taken from:

Software-Related Recalls: An Analysis of Records, The Biomedical Instrumentation & Technology journal

Lisa K. Simone (*biomedical and software engineer with the Center for Devices and Radiological Health at the U.S. Food and Drug Administration*)

Year	Total Recalls	Software-Related Recalls	Percent
2005	604	84	13.9%
2006	663	119	17.9%
2007	638	119	18.7%
2008	847	192	22.7%
2009	782	146	18.7%
2010	981	147	15.0%
2011	1,277	315	24.7%

Percentage of Recalls Related to Software

Z. Fu, C. Guo, S. Ren, Y. Jiang and L. Sha, "Study of Software-Related Causes in the FDA Medical Device Recalls," *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*, Fukuoka, 2017, pp. 60-69

2. PVSio-web framework

PVSio-web

PVSio-web [1, 2] is an open-source environment enabling developers and users of medical devices to assess and validate them with respect to human-machine interaction.

This makes it possible to discover subtle interfacing errors that may have grave consequences, such as under- or over-dosing [3].

PVSio-web is implemented in JavaScript by a software platform composed of several scripts, invoked and coordinated through a web interface.

1. Oladimeji, P., Masci, P., Curzon, P., Thimbleby, H.: PVSio-web: a tool for rapid prototyping device user interfaces in PVS. In: FMIS2013, 5th International Workshop on Formal Methods for Interactive Systems (2013)
2. Masci, P., Oladimeji, P., Curzon, P., Thimbleby, H.: PVSio-web 2.0: Joining PVS to Human-Computer Interaction. In: 27th International Conference on Computer Aided Verification (CAV2015). Springer (2015)
3. Masci, P., Ruks_enas, R., Oladimeji, P., Cauchi, A., Gimblett, A., Li, Y., Curzon, P., Thimbleby, H.: The benefits of formalising design guidelines: A case study on the predictability of drug infusion pumps. *Innov. Syst. Softw. Eng.* 11(2), 73-93, 2015

PVSio-web

The tool and application examples available at
<http://www.pvsioweb.org>

The screenshot shows the PVSio-web website interface. At the top, there is a navigation bar with links: [HOME](#), [LIVE VERSION](#), [DOWNLOADS](#), [PUBLICATIONS](#), [VERIFICATION TOOLS](#), [RELATED TOOLS](#), and [DOWNLOAD STATS](#). Below the navigation bar, the main heading is "PVSio-web" followed by the subtitle "A formal methods toolkit for model-based development of human-machine interfaces." The central part of the page features a diagram of a human-machine interface (HMI) with a central display area and several control buttons labeled "Ok", "Menu", "Start", and "On Off". Below the diagram, there is a code editor window showing a snippet of code with a search bar and navigation controls. The code includes a definition for a type and a state declaration. On the right side, there is a sidebar with a search bar and a list of application examples: GraphBuilder, SafetyTest, Interactions recorder, and PVSio-web console.

PVSio-web

PVSio-web consists of two tools:



PVSio-web uses SRI's state-of-the-art theorem prover PVS (Prototype Verification System) [4] for analysis, and the PVSio component [5] as a basis for simulation.

To install PVSio-web, first you need to install PVS and NodeJS, and then clone the PVSio-web github repository.

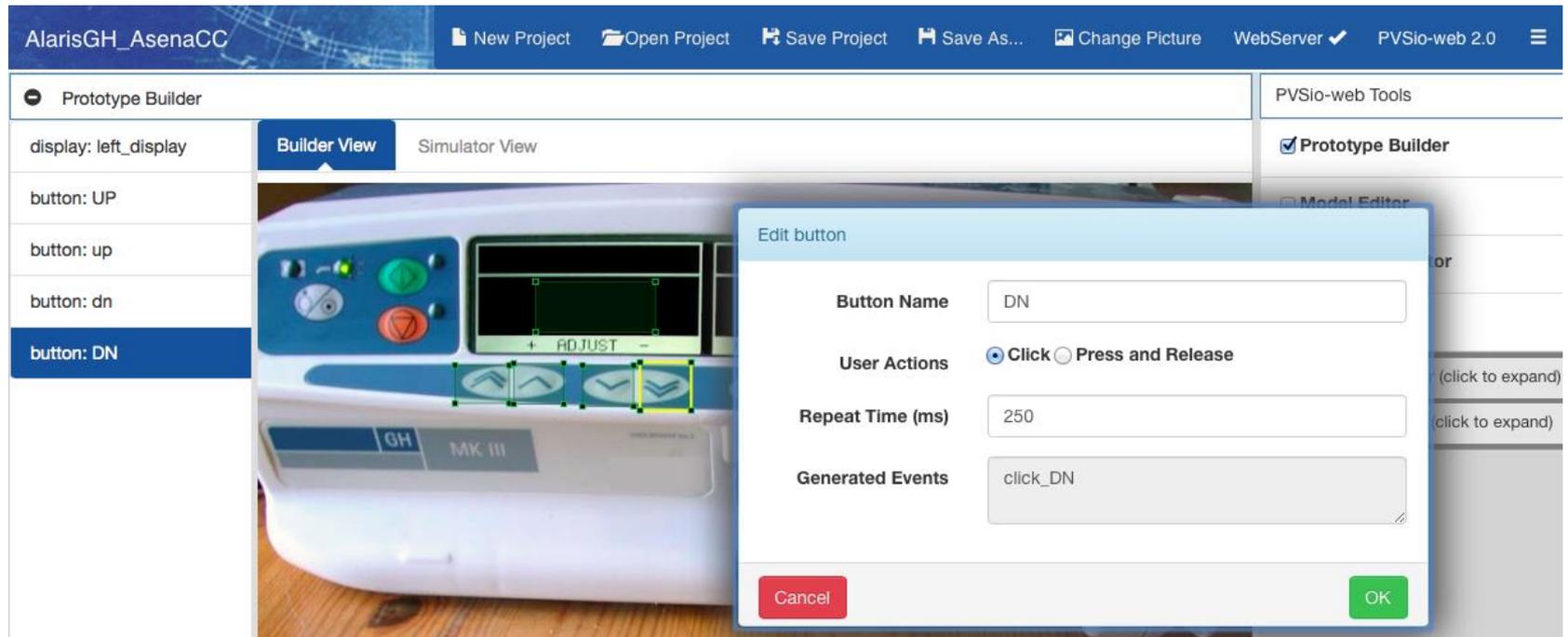
- PVS is open source, under the GNU General Public License (GPL), <http://pvs.csl.sri.com/download.shtml>.
- NodeJS can be downloaded at <http://nodejs.org>

4. S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas, "PVS: combining specification, proof checking, and model checking," in *Computer-Aided Verification, CAV '96*, ser. LNCS, R. Alur and T. Henzinger, Eds. Springer-Verlag, 1996, no. 1102, pp. 411–414.

5. C. Muñoz, "Rapid prototyping in PVS," National Institute of Aerospace, Hampton, VA, USA, Tech. Rep. NIA 2003-03, NASA/CR-2003-212418, 2003.

The Prototype Builder tool

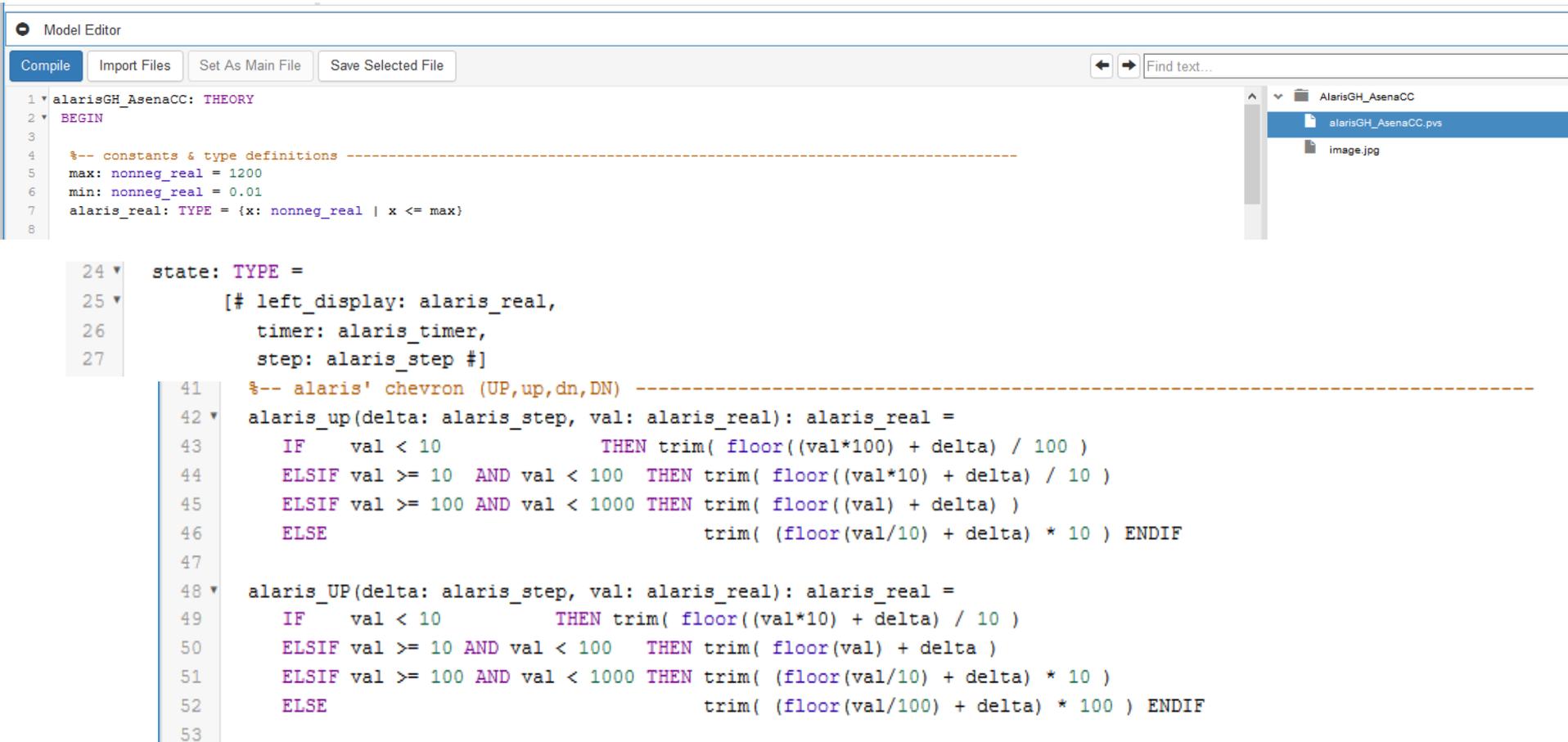
A developer uses the Prototype Builder tool to create a graphical representation of the device's front panel and link its controls and displays to functions describing how the device responds to user actions on the controls and how it shows information on the displays.



Screenshot of Builder View section

The Prototype Builder tool

A developer uses the Model Editor to create the system model



The screenshot shows the Model Editor interface. The main window contains a code editor with the following code:

```
1 alarisGH_AsenaCC: THEORY
2 BEGIN
3
4  %-- constants & type definitions -----
5  max: nonneg_real = 1200
6  min: nonneg_real = 0.01
7  alaris_real: TYPE = {x: nonneg_real | x <= max}
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24 state: TYPE =
25     [# left_display: alaris_real,
26      timer: alaris_timer,
27      step: alaris_step #]
28
29
30
31
32
33
34
35
36
37
38
39
40
41  %-- alaris' chevron (UP,up,dn,DN) -----
42  alaris_up(delta: alaris_step, val: alaris_real): alaris_real =
43      IF    val < 10                THEN trim( floor((val*100) + delta) / 100 )
44      ELSIF val >= 10 AND val < 100 THEN trim( floor((val*10) + delta) / 10 )
45      ELSIF val >= 100 AND val < 1000 THEN trim( floor((val) + delta) )
46      ELSE
47          trim( (floor(val/10) + delta) * 10 ) ENDIF
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

The right sidebar shows a file explorer with the following files:

- AlarisGH_AsenaCC
- alisGH_AsenaCC.pvs
- image.jpg

The Simulation Environment

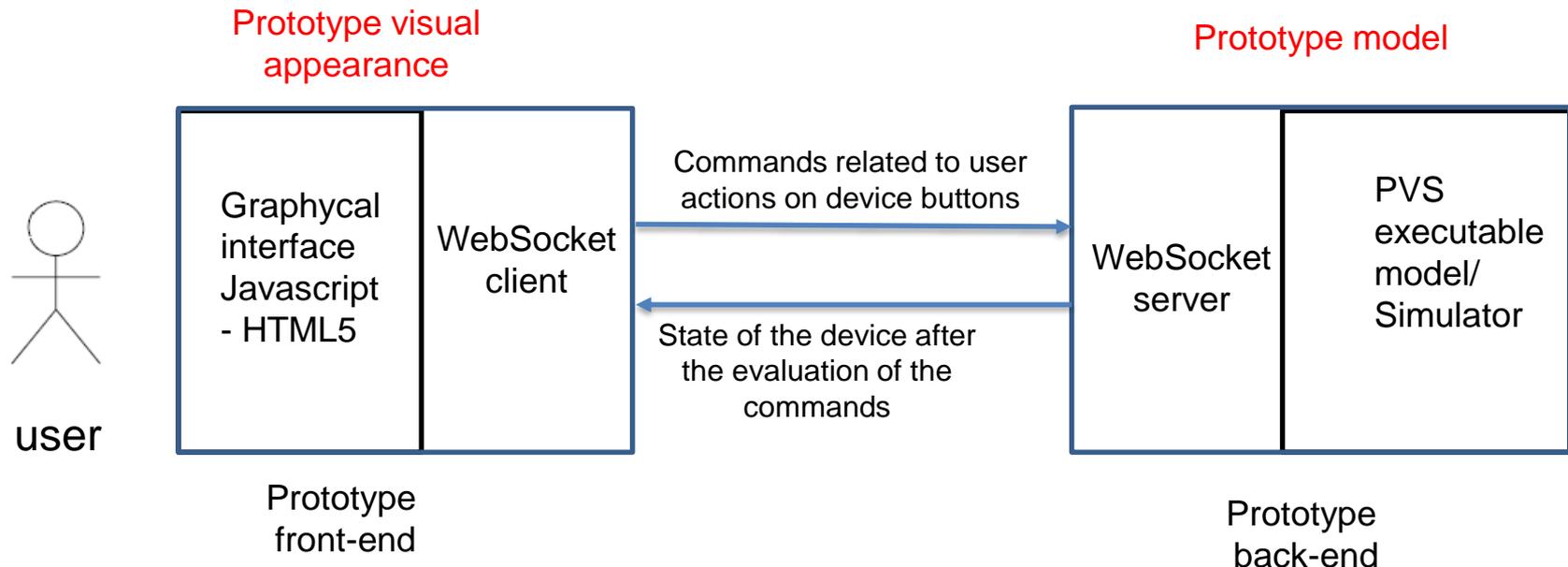
A Simulation Environment enables a developer or a prospective user to simulate the device and interact with the simulation through the image of its panel.



Screenshot of Simulator View section when the user clicks on a virtual device button (the up arrow key or the double up arrow key) several times and reach 0.27 value and the each of this action is shown on virtual device display.

Behind PVsio-web

A PVsio-web prototype follows a Model-View-Controller (MVD) design pattern, which promotes a clear separation between the behavior of the prototype and its visual appearance.



Prototype front-end

Basic elements for the visual appearance of the prototype:

- a realistic picture of the device
- a Javascript module that
 - contains the interactive widgets,
 - captures user interactions with the prototype and
 - renders the device state
- The interactive widgets can be
 - input widget (e.g., button, sliders, ...)
 - output widget (e.g., digital displays)
- a HTML file, for loading and executing the visual appearance of the prototype in a web browser

PVSio-web widget library

The widget library (written in Javascript) can be used for

1. capturing user actions performed on selected regions of the prototype
2. animating selected regions of the prototype in response to user actions
3. Translating user actions into commands that drive the evaluation of the device model simulated on the PVSio-web back-end

Example: DISPLAY widget, using Basic Display (BasicDisplayEVO)

instantiate the widget using the constructor method which takes three arguments

- A unique identifier for the widget
- The position and size of the widget (top, left, high, width)
- Optional attribute for customizing the visual aspects and functionality of the widget
- use the *render* method to make the widget visible and render, for example, the string “Hello world”

PVSio-web widget library: Basic Display



```
1. require.config({
2.     baseUrl: "../../client/app",
3.     paths: { d3: "../lib/d3", text: "../lib/text" }
4. });
5. require([ "widgets/core/BasicDisplayEVO" ], function (BasicDisplay) {
6.     "use strict";
7.     var disp = new BasicDisplay("disp", {
8.         top: 62, left: 44, height: 410, width: 294
9.     }, {
10.        fontSize: 12, backgroundColor: "steelblue"
11.    });
12.    disp.render("Hello World!");
13. });
```

To execute the prototype

1. start the PVSio-web back-end
2. open a web browser at the page
<http://www.pvsioweb.org/tutorials/HelloWorld>

Loading and executing the prototype

HTML code for loading the javascript file of the prototype in the web browser

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta charset="utf-8">
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
6.     <title></title>
7.     <meta name="description" content="">
8.     <meta name="viewport" content="width=device-width">
9.     <link href="../../client/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" media="screen">
10.    <link href="../../client/css/style.css" rel="stylesheet" type="text/css"/>
11.    <link href="../../client/css/animate.css" rel="stylesheet" type="text/css"/>
12.    <link href="../../client/lib/jquery-toggles/css/toggles.css" rel="stylesheet">
13.    <link href="../../client/lib/jquery-toggles/css/themes/toggles-modern.css" rel="stylesheet">
14.    <link href="../../client/lib/bootstrap/css/bootstrap-slider.css" rel="stylesheet" type="text/css"/>
15.  </head>
16.  <body class="noselect" style="background:#dedfdd;">
17.    <div id="content" class="content">
18.      
19.      <div id="screen">
20.
21.      <script defer="true" src="../../client/lib/layout.js"></script>
22.      <script defer="true" src="../../client/lib/promise-0.1.1.js"></script>
23.      <script defer="true" src="../../client/lib/keys.js"></script>
24.      <script defer="true" src="../../client/lib/jquery.js"></script>
25.      <script defer="true" src="../../client/lib/underscore/underscore.js"></script>
26.      <script defer="true" src="../../client/lib/d3/d3.js"></script>
27.      <script defer="true" src="../../client/lib/backbone.js"></script>
28.      <script defer="true" src="../../client/lib/handlebars-v1.3.0.js"></script>
29.      <script defer="true" src="../../client/lib/bootstrap/js/bootstrap.min.js"></script>
30.      <script defer="true" src="../../client/lib/jquery-toggles/toggles.min.js"></script>
31.      <script defer="true" src="../../client/lib/bootstrap/js/bootstrap-slider.js"></script>
32.      <script defer="true" src="../../client/require.js" data-main="index.js"></script>
33.    </div>
34.  </body>
</html>
```

Prototype back-end

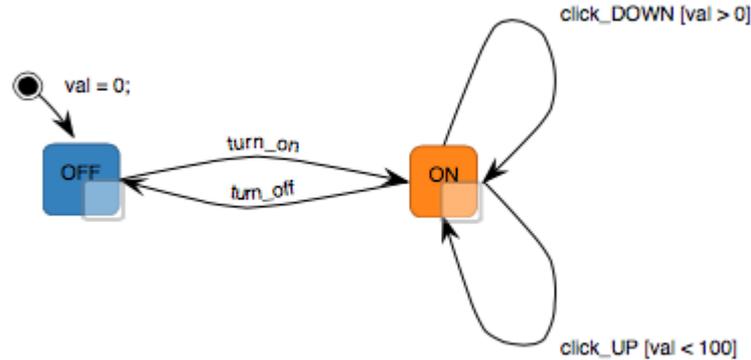
The model of the behaviour of the system must be written in the PVS language. It can be entered textually with the **Model Editor**, but it can also be generated from the **Emucharts Editor**.

Emucharts is a graphical language to define finite-state automata extended with a set of variables, and where each transition is annotated with an event trigger, an optional guard, and an optional action.

The event trigger models an atomic input to the automaton, the guard is a logical condition (true by default) on the variables, and the action is a set of assignments to the variables.

An Emucharts automaton is translated into an executable PVS theory by the PVSio-web Code Generator.

The Emuchart language



A simple example is the following: a two-state diagram where turning on the device changes its state from OFF to ON and only in the ON state it is possible to click buttons UP or DOWN (both of clicks are bounded by the minimum and maximum thresholds 0 and 100 values).

val: integer; initialised at 0; min = 0; max = 100

ON, OFF: button

UP, DOWN: button

User action: click button ON, click button OFF,
click button UP, click button DOWN

How to build a prototype: a simple example

Download and install PVSio-web on your computer

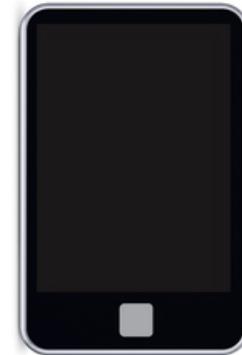
- Start PVSio-web back-end

 - `./start.sh`

- Create a New Project

 - insert the name

 - load picture: `tutorials/Echo/device.png`



Prototype Builder:

- Create Numeric Display. Name: **d**
- Create a Button. Name: **b**



How to build a prototype: a simple example

ModelEditor:

- write the model or import a file with the model

Simulator View

- simulate the model

```
counter:THEORY
BEGIN
  state:TYPE =[#
    d:integer #]

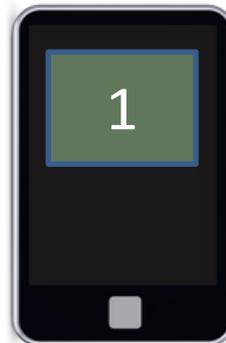
  init (x:real):state =(# d := 0 #)

  click_b (st:state) :state =
  st WITH =[# d := d(st)+1 #]

  tick(st:state):state =st
END counter
```



after 1click



To sum it up

- 1) Take a picture of the user interface of the real system that could be used as a basis to create the visual appearance of the interactive simulation, the front-end of the prototype.
- 2) Build an executable model (an executable specification) of the behavior of the system in the PVS language, the back-end of the prototype.
- 3) Use the PVSio-web library to create interactive widgets over the picture of the system:
 - Input widgets translate user actions over buttons into expressions of the executable PVS model to be evaluated to compute the system response
 - Output widgets mirror state attributes of the PVS model and resemble the look & feel of the real system in the corresponding state.
- 4) Use the simulator to validate the user interface of medical devices by interactively animating a formal specification of the user interaction with the device.

Demos

The following demos are available in the PVSio-web framework
To execute a demo, open a web browser at the specified page

BBraun perfusor

<http://www.pvsioweb.org/demos/BBraun>



Radical7 Patient monitor

<http://www.pvsioweb.org/demos/Radical7>



Stellant contrast media injector

<http://www.pvsioweb.org/demos/stellantV2>



AlarisGP infusion pump

<http://www.pvsioweb.org/demos/AlarisGP>



3. A case study on infusion pumps

Infusion pumps

Infusion pumps are medical device that deliver drugs and nutrients into a patient body at controlled rates and volumes

Rate and volumes are usually entered by nurses using buttons and keys on the device interface

Design errors that can lead to situations in which wrong rates or volumes can be accidentally inserted in the device have been identified.

Work developed within the CHI+MED research project (<http://www.chi-med.ac.uk/>), and in collaboration with the Center for Devices and Radiological Health of the US Food and Drug Administration (CDRH/FDA).

CHI+MED (Computer-Human Interaction for Medical Devices, *EP/G059063/1*) was an EPSRC-funded project to improve the safety of interactive (programmable) medical devices.

EPSRC - *Engineering and Physical Sciences Research Council* (UK's agency for funding research in engineering and the physical sciences)

Bbraun Perfusor



Software model of the device
Greyed out keys are disabled
in this simulation

An example of user interface with navigation keys: 4 arrow keys and a display

In the manual: to enter volume or rates, just use the arrow keys (up key/down key, to increment/decrement the digit; left/right key to select different digits)

Digits are not independent.

Issues when we enter high value: unexpected behaviour of the device.

Zimed Syringe pump



Another example of user interface with navigation keys:
4 arrow keys and a display

Independent digits.

We are allowed to go from the rightmost position to the leftmost position with one click.

Arcomedical Syramed pump



Another user interface with navigation keys.

Independent UP and DOWN keys, and the impression is that each key operates on a different digit.

But ... digits are not independent. Moreover when you overshoot the maximum value, the display shift numbers, and the last integer digit is smaller than the other integer digits.

Baxter Colleague



The device ignores key presses: 20.01 is displayed 20.0 (for numbers above 10, only one fractional digit: for numbers above 100, no fractional digit)

The device ignores the decimal point: 214.2 returns High value (2142)

But 100.1 is displayed as 1001, and the infusion can be started

Viewing angle in seven segment display



Patient monitors
Alcon Everest
Datascope Accutorr Plus



Alaris PC
infusion pump



Lessons learned

Identification of design issues

- user input erroneously discarded
 - inappropriate feedback
 - unexpected device modes
 - some device silently discard after a timeout
 - other devices silently confirm after a timeout
-

A recorded video of the demonstration is available on YouTube
"Medical Device Training - Design Issues in Medical User Interfaces"

<https://www.youtube.com/watch?v=T0QmUe0bwL8>

4. Building a prototype of a contrast media injector

joint work with

Paolo Masci, Department de Informatica, Universidade do Minho, Braga, Portugal

Davide Caramella, Ruggero Dell'Osso, Department of Diagnostic and Interventional Radiology, University of Pisa, Italy

Bernardeschi C, Masci P, Caramella D, Dell'Osso R.

The benefits of using interactive device simulations as training material for clinicians: an experience report with a contrast media injector used in CT. Medical Cyber Physical Systems Workshop 2018, Porto. SIGBED Review newsletter (2018, to appear)

Stellant CT contrast media injector

Contrast media injectors inject contrast and saline into the bloodstream of patients. Iodinated media can be nephrotoxic, being a known cause of possible acute renal failure in hospitalised patients

To minimize the risk of adverse health problems, it is therefore important to set up the injectors so that it delivers the minimal amount of contrast media necessary for the diagnostic task

Sophisticated interfaces and injection setting are available

The simulation

- was created to support training of clinicians, and
- helped to identify and raise awareness among clinicians of critical workflows that could induce accidental use errors that may have safety implications

Stellant CT contrast media injector



Commonly used in CT in hospitals

The injection system includes a workstation and an injector

Interaction with the workstation is carried out through a touchscreen display

- Dual-syringe injector that performs injection of contrast and saline into the bloodstream of patients
- The workstation allows clinicians to set up and manage personalized injection protocols for different patients, based on parameters such as patient weight, past scan procedures, current scan settings, diagnostic tasks, etc.

The injector

blue for saline green for contrast



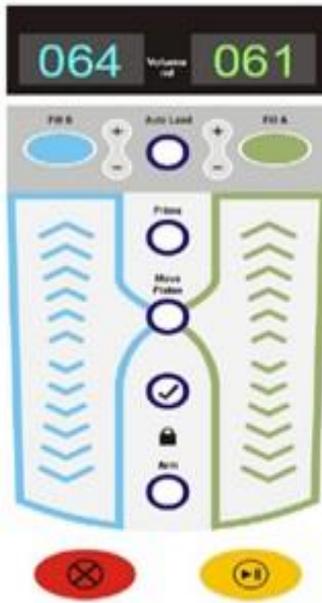
The front panel of the injector uses a number of displays and LEDs to provide feedback to the clinician about the state of the device.

Various buttons on the front panel of the injector allow clinicians to operate the device.

The body of the device includes an injector head (where syringes are inserted), plungers for controlling the volume of liquid in the syringes (plungers are automatically advanced and retracted when syringes are inserted and removed), and tubes/needles (used to connect syringes and the patient).

The workstation seamlessly communicates with the injector, allowing clinicians to monitor progress of the injection directly from the workstation screen.

The injector: displays and lights

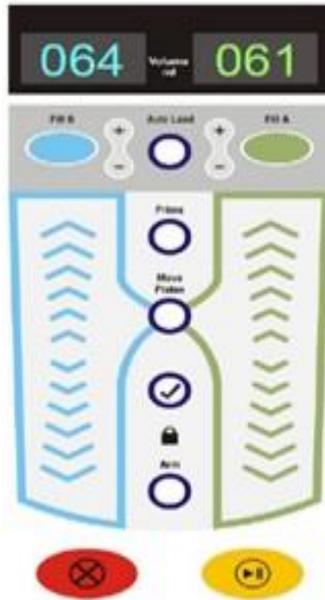


- Two seven-segments displays (VolumeA and VolumeB) report, depending on the device mode, either the volume of liquid in the syringe, or the position of the plunger.

Each display has three significant digits, and can render only integer numbers.

- One LED light placed next to a lock symbol indicates whether the injection protocol has been set and locked from the workstation.
- Two large LED lights indicate whether an injection is running.

The injector: filling syringes



Automatic filling

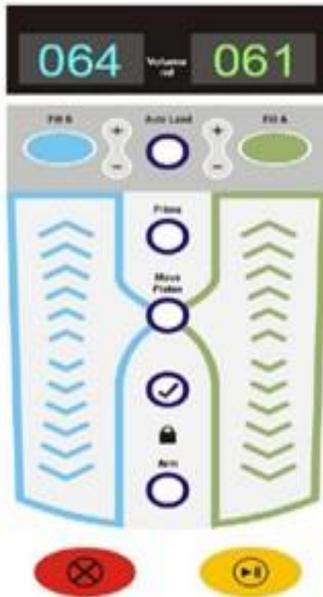
An Autoload button, can be used to move the plungers and load the volume of saline and contrast configured by the clinician on the workstation (plus (+) and minus(-) can be used to increase/decrease the volume before filling syringes).

Two buttons FillA and FillB activate the autoload sequence for the syringes.

Manual filling

A Manual load button enables manual adjustment of the plunger position using the chevron keys available on the front panel of the injector (speed - where the chevron keys are pressed)

The injector: control injection



A Prime button can be used to remove air-in-line

A Check-Air button is for checking air-in-line

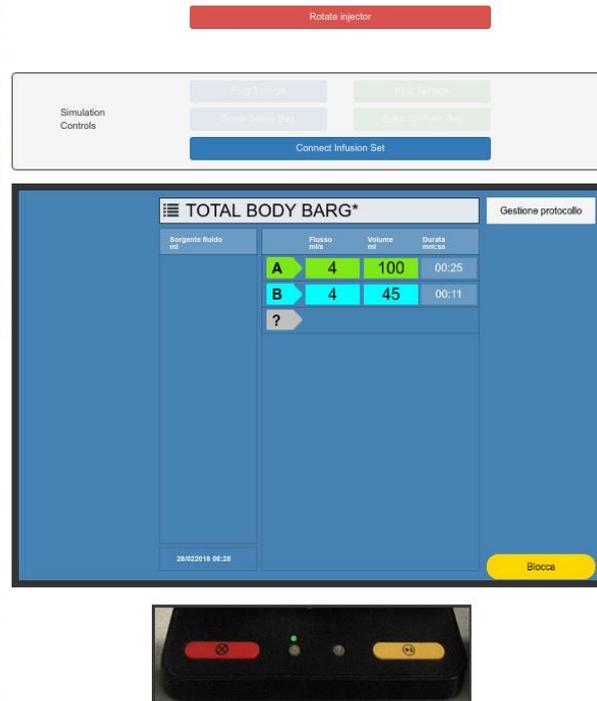
An Arm button is used to make the system ready for an injection.

An Abort button terminates an injection procedure and disarms the injection.

A Start/Hold button allows to start the injection (when the injection is not started), and to pause the injection (this function is active when an injection is running).

The contrast media injector prototype

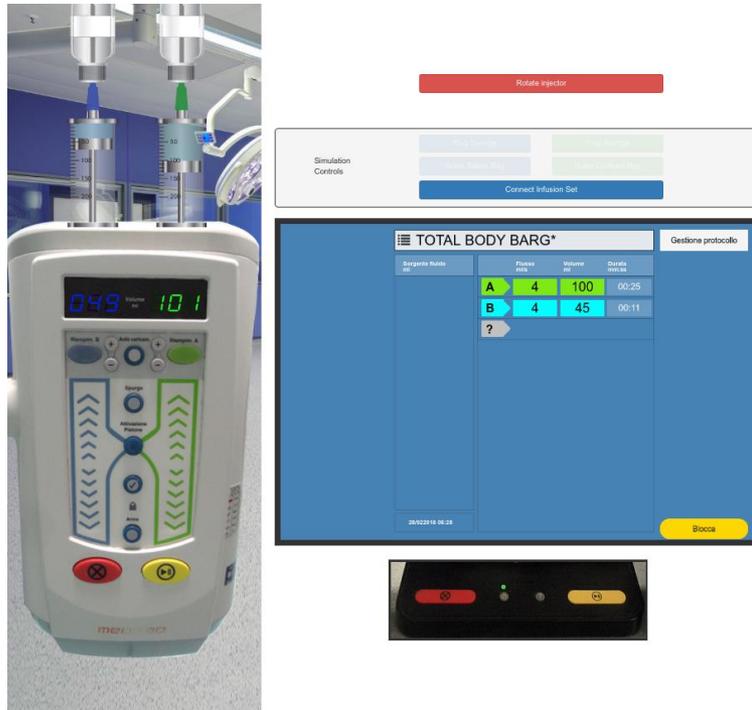
an interactive simulation of the complete injection system using the PVSio-web prototyping framework



screenshot of the simulation

Live version of the simulation: <http://www.pvsioweb.org/demos/stellantV2>

Development of the interactive simulation



In the example, output widgets are used to represent a system state where

- the syringes are plugged into the injector and spiked to a bag with saline and contrast liquids
- the injector has completed the process of loading the saline and contrast liquids in the syringes (the two seven segments displays on the front panel of the injector indicate the volume of liquid loaded in the two syringes)

Development of the prototype

The specification of the system is reverse engineered using in combination

- the user manual
- direct interaction with the real device
- the results of a field study we conducted that focused on how expert users routinely operate the device

This allowed the entire team to look closely and in a systematic manner into various design aspects of the system.

This greatly helped engineers understand how clinicians use the system, and greatly helped the entire team to discuss and demonstrate various corner cases that could potentially have safety consequences in specific contexts.

Executable PVS specification

Two main steps:

Specify the system state as a PVS record type with relevant state attributes

```
state: TYPE = [#  
  mode: Mode,  
  vol_saline: Volume,  
  vol_contrast: Volume,  
  lock_LED: LED,  
  ...  
#]
```

Specify the behavior of the system as a set of transition functions that range over system states.

```
1 click_btn_manual(st: state): state =  
2   st WITH [  
3     mode := MANUAL,  
4     vol_saline := plunger_saline(st),  
5     vol_contrast := plunger_contrast(st),  
6     vol_saline_confirmed := FALSE,  
7     vol_contrast_confirmed := FALSE,  
8     btn_manual_timeout := BTN_MANUAL_TIMEOUT  
9   ]
```

Executable PVS specification of the injection system

The size of the PVS specification is approx. 800 lines.

The PVS record type for the injector system includes 57 state attributes

- 38 attributes for the injector state
- 14 attributes for the workstation state
- 5 attributes for the state of the syringes

The PVS specification includes 33 transition functions

- 26 for modelling the behavior of the injector
- 7 for modelling the workstation

The main focus of the simulation was the injector: this is the reason behind the small number of transition functions used for modelling the workstation

Executable PVS specification of the injection system

The PVS specification includes the following PVSio-web widgets:

- 33 buttons
- 9 displays
- 5 LEDs
- 2 syringes

Each button widget is linked to a transition function in the PVS model: this is done through the APIs of the PVSio-web widget, which include a parameter for specifying the name of the transition function to be evaluated when a given user action is performed on the widget.

Each display and LED widget is seamlessly associated with a state attribute defined in the PVS specification. The creation of these widgets follows a pattern that is similar to that for button widgets.

Example of button widget

Manual load button widget

```
var sys = {};  
sys.btn_manual = new Button("btn_manual", {  
  top:792, left:210, width:38, height:38  
}, {  
  callback: render  
});
```

Button is the widget constructor

The first argument of the constructor is a string defining the widget identifier.

The created widget is stored in a field `btn_manual` of a variable `sys`.

Transition function in the PVS model to be linked to the widget is constructed by concatenating the user action that activates the widget with the widget identifier (e.g., `click_btn_manual` when the user clicks on the button)

The second argument defines the coordinates and size of the widget

The third argument provides information about which callback function is to be invoked for refreshing the visual appearance of the prototype

Example of display widget

LED widget

```
sys.lock_LED = new LED("lock_LED", {  
  top:916, left:221, width:13, height:13  
}, {  
  color: "green"  
});
```

LED is the widget constructor

The first argument is the widget identifier

The second argument defines position and size of the widget

The third argument specifies the LED color

Refresh of the visual aspects of widgets

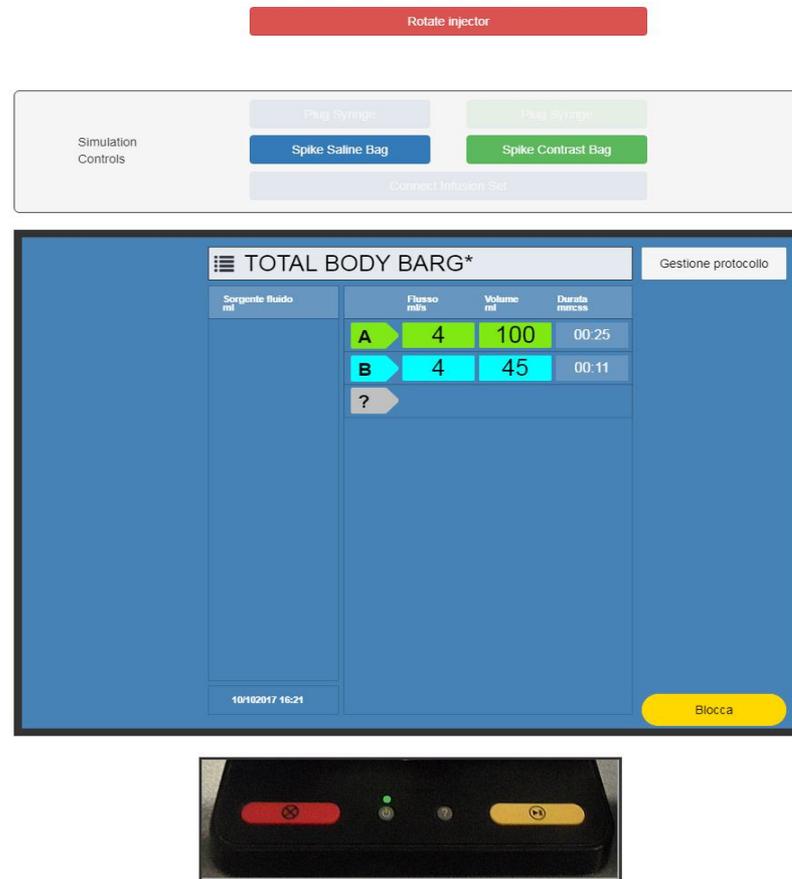
The visual aspect of all widgets is periodically refreshed every time the PVS specification is evaluated.

The evaluation of the specification occurs either when the user interacts with an input widget (e.g., presses a button), or periodically (if the device has internal timers that are ticking). A JavaScript function `render` contains the code for refreshing the widgets.

```
function render(err, event) {  
  var res = stateParser.parse(event.data);  
  if (res) {  
    sys.btn_manual.render(res);  
    sys.lock_LED.render(res);  
    ...  
  }  
}
```

In its basic form, the `render` function simply parses the PVS state and invokes the `render` method of the widgets.

Stellant CT Injection System: prototype



Software model of the device and the console

<http://www.pvsioweb.org/demos/stellantV2>

Risk of incorrect injection settings

The Volume displays normally report a value corresponding to the volume of liquid loaded in the syringes.

However, in certain operating modes for the injector, the display values have a different meaning.

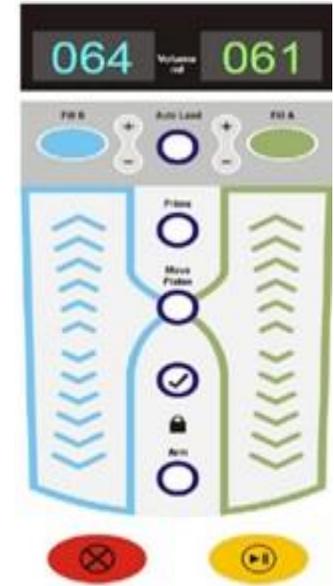
Information on the front panel of the injector is not always sufficient to discriminate these different cases.

Syringes are not plugged, the display values indicate the maximum value of volume that can be loaded in the syringes;

Syringes are connected but empty, the display values indicate the current position of the syringe plungers;

Button AutoFill on the front panel is pressed, the display values indicate the target volume of liquid that will be loaded in the syringes.

This value is reset to 0 as soon as buttons FillA or FillB are pressed.



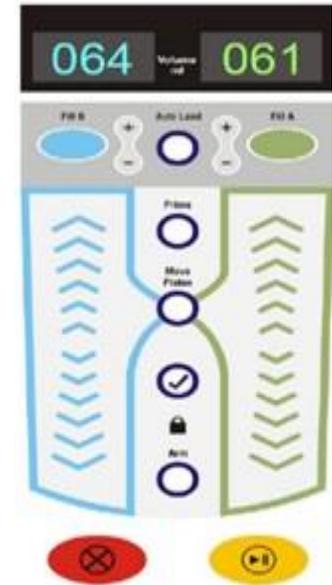
Risk of undetected air-in-line

For patient safety, it is important to check that air bubbles are purged from syringes and tubing before the injection.

For this reason, the arming phase of the injector is disabled if the CheckAir button available on the front panel of the injector has not been pressed.

However, this button is only a placeholder, i.e., a functionality provided by the device to remind the clinician to verify the absence of air (the injector does not have any sensor for detecting air-in-line).

Pushing the CheckAir button does not trigger any actual check from the device – the clinician needs to look into syringes and tubes and make sure there are no air bubbles. If air bubbles are present, the clinician can use the Prime button to remove the air.



Risk of misprogramming the injector

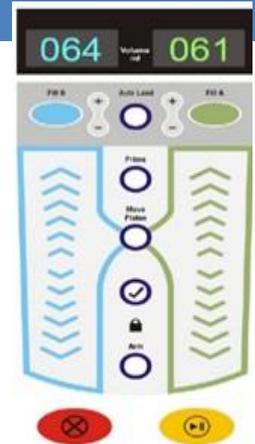
The injector provides two main modalities for filling syringes, and two modalities for priming: automatic and manual. These two modalities can be *interleaved*.

Manual mode: clinicians use the chevron keys to load the volume prescribed by the protocol and prime the syringes.

Automatic mode, a single button press on FillA and FillB on the front panel of the injector allows clinicians to load liquid in each syringe, and then a single button press on the Prime button primes the syringes.

The volume of liquid loaded using automatic mode is larger than the volume prescribed by the protocol: +h mL for the contrast, and +k mL for the saline. The automatic prime function pushes exactly h mL of contrast and k mL of saline out of the syringes.

If clinicians interleave the two modalities and accidentally omit to check the volume on the injector display, there is a risk of injecting a volume of saline and contrast that is slightly different than the intended values.



Risk of misreading values

This issue concerns the phase in which the injector needs to be connected to the patient to start the injection.

In this phase, the injector needs to be rotated of 180 degrees. The rotation moves air up in the syringes – this is a safety precaution for preventing air being injected in the veins of the patient.



However, the rotation causes the displays provided on the front panel of the injector to be upside-down. This is particularly unfortunate, because the device uses seven-segments displays and certain numbers can be accidentally misread when the display is upside-down (e.g., 51 can be misread as 12, a well-known problem with using seven-segments displays in medical devices).

Summing it up

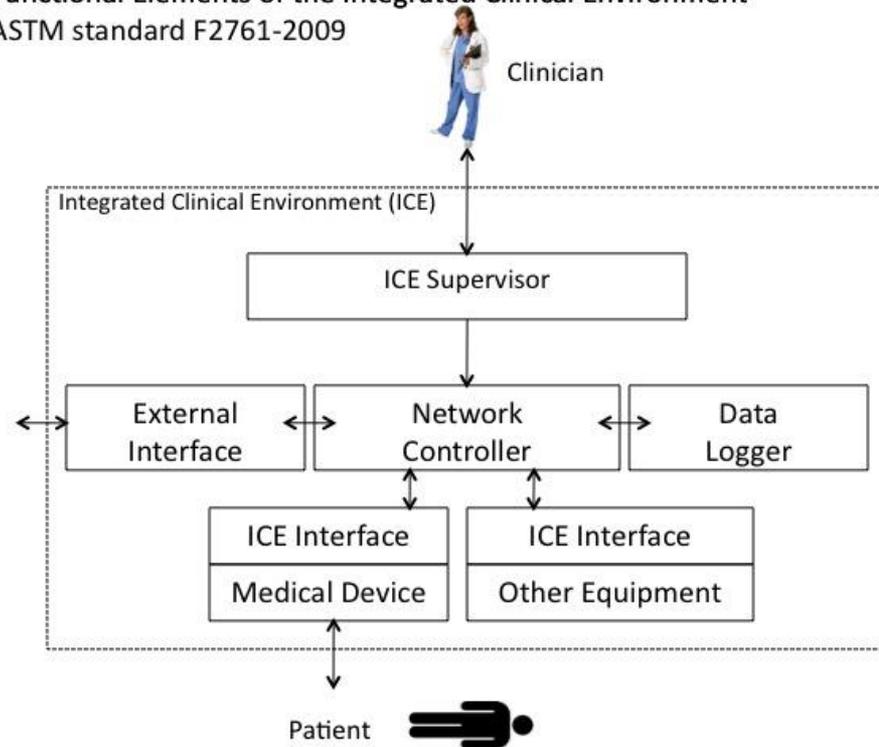
- The interactive simulation stimulated a constructive discussion within a multidisciplinary team of engineers and clinicians, about possible design improvements to the device that could prevent the identified critical workflows.
- Clinicians played a fundamental role in the identification of critical scenarios, as well as in the description of how the medical device is routinely used in the real-world.
- A possible use of the simulation tool is to enhance the proficiency of the clinical users of the injector, helping them to avoid possible traps, thus increasing patient safety.

The simulation facilitated the multidisciplinary work necessary to obtain results that have strong impact and immediate utility to different stakeholders.

5. Prototypes of more complex systems

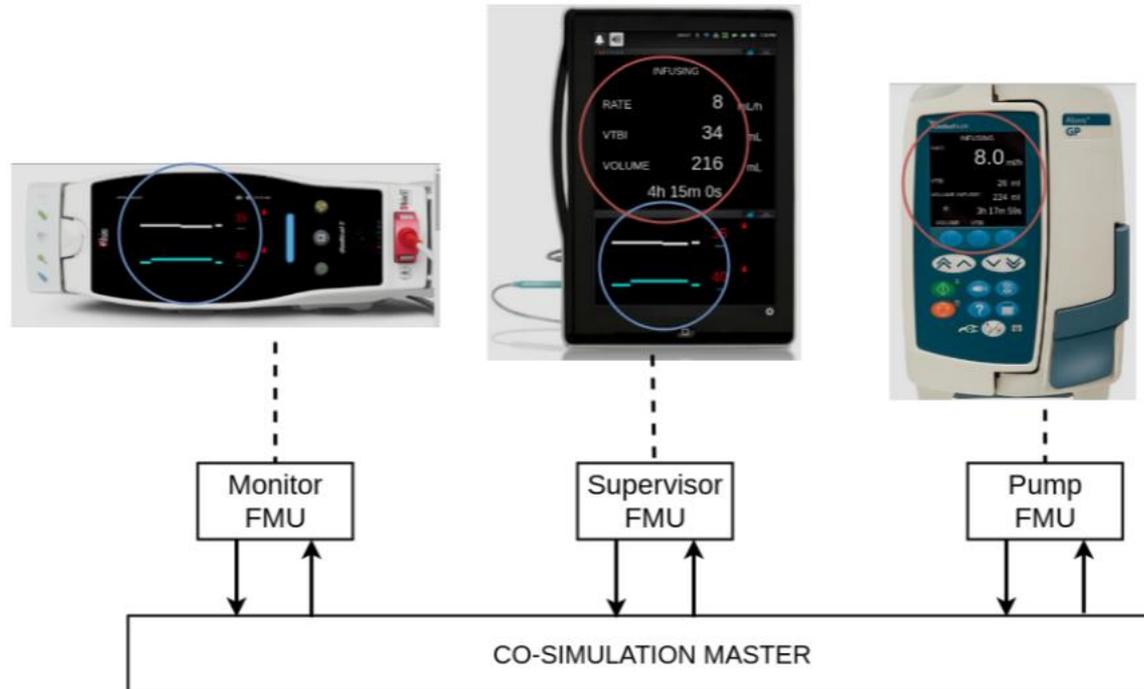
The case of Integrating Clinical Environments

Functional Elements of the Integrated Clinical Environment
ASTM standard F2761-2009



ICE

PVSio-web has been extended to introduce support of automatic generation of user interface prototypes equipped with a standard FMI co-simulation interface.

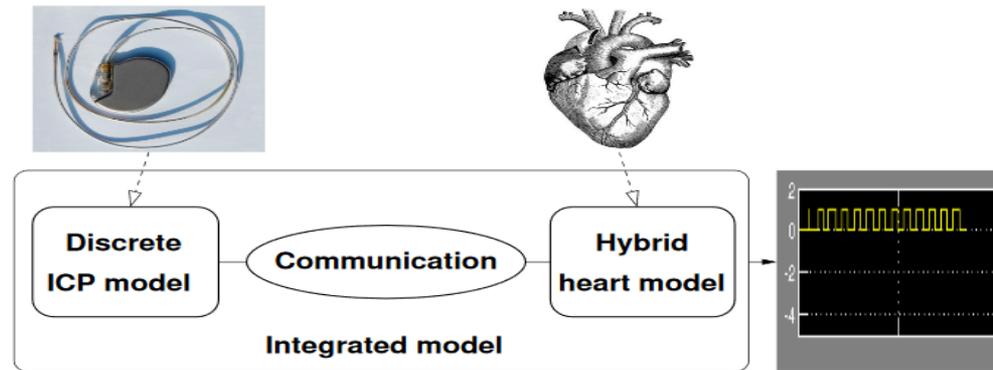


M. Palmieri, C. Bernardeschi, P. Masci, A Flexible Framework for FMI-based Co-Simulation of Human-Centred Cyber-Physical Systems, 2nd Workshop on Formal Co-Simulation of Cyber-Physical Systems, a satellite event of SEFM2018, June 26, 2018, Toulouse, France

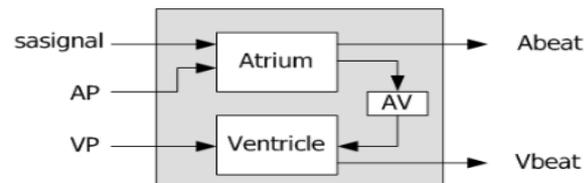
Heart-pacemaker prototype

Cinzia Bernardeschi, Andrea Domenici, Paolo Masci:

A PVS-Simulink Integrated Environment for Model-Based Analysis of Cyber-Physical Systems. IEEE Trans. Software Eng. 44(6): 512-533 (2018)

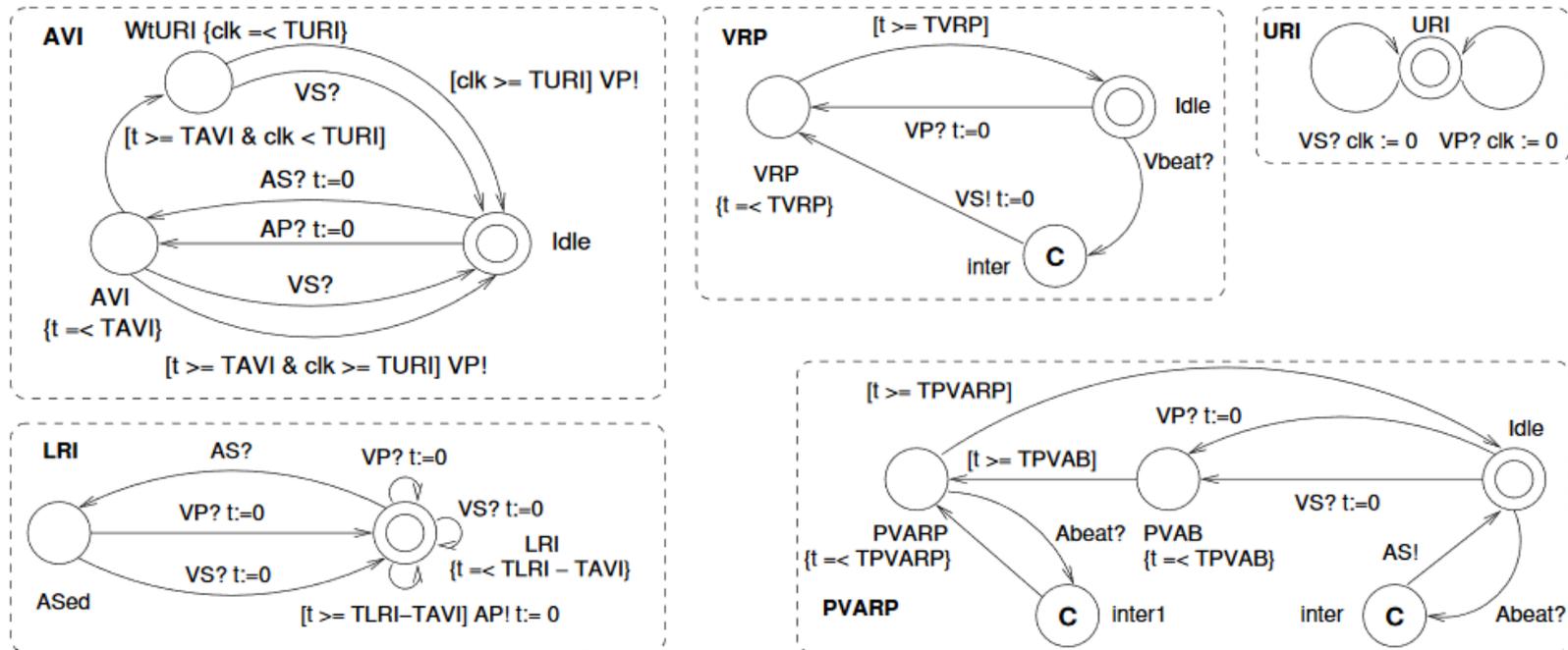


Architecture of the heart model (Chen et al., 2014)



(Chen et al., 2014) T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre, Quantitative verification of implantable cardiac pacemakers over hybrid heart models, Information and Computation, vol. 236, n. 0, 2014.

The pacemaker model as a network of Timed Automata

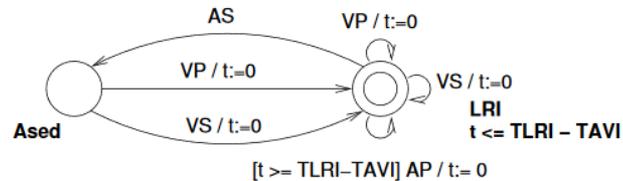


(Jiang et al., 2012) Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam, Modeling and verification of a dual chamber implantable pacemaker, in Tools and Algorithms for the Construction and Analysis of Systems, LNCS vol. 7214, 2012.

From Timed Automata to PVS

Modelling patterns have been defined to represent TAs in PVS executable specifications.

```
LRI: THEORY BEGIN
Mode: TYPE = {LRI, ASed}
state: TYPE = [#
    time: real,
    loc: Mode #]
```



```
init_LRI: state = (# time := 0, loc := LRI #)
```

```
en_APout(st: state): boolean = % enabling
    loc(st) = LRI AND time(st) >= TLRI-TAVI
```

```
APout(st: (en_APout)): state = (# time := 0, loc := LRI #)
```

.....

```
en_tau(st: state): bool = false % time-checking predicate
```

```
tau(st: state): state = st % timing function
```

```
END LRI
```

The prototype

The PVSio-web co-simulation environment integrates the two models:

- The Heart model simulation is started in the Simulink environment
- the PVSio environment is loaded with the PVS theory describing the Pacemaker
- PVSio-web requests the heart model to return heartbeat signals
- PVSio-web sends PVSio a function call with heartbeat signals as arguments
- PVSio computes the pacemaker model response and returns it to PVSio-web
- PVSio-web sends the pacemaker response to the heart model

Simulation

Debugger Monitor

PVSio WebSocket

```
#),          vrp:=(# loc:=Idle,
time:=29 #) #),          Vget:=0,          VP:=0,
vpon:=FALSE,          vptime:=0,          wclk:=29 #)

<-RECEIVED
[ 10:23:48 GMT+0200 (CEST) ]
(# Aget:=0,          AP:=0,          apon:=FALSE,
aptime:=0,          dev:=(# avi:=(# clk:=29,
loc:=Idle,          time:=30 #),
lri:=(# loc:=LRI,          time:=30 #),
pvarp:=(# loc:=Idle,          time:=30
#),          uri:=(# clk:=30
#),          vrp:=(# loc:=Idle,
time:=30 #) #),          Vget:=0,          VP:=0,
vpon:=FALSE,          vptime:=0,          wclk:=30 #)
```

Simulink WebSocket

```
(# Aget:= 0.00, Vget:= 0.00 #)
<-RECEIVED
(# Aget:= 0.00, Vget:= 0.00 #)
```

From prototyping to formal verification



The *Prototype Verification System* is an interactive theorem prover developed at SRI International by S. Owre, N. Shankar, J. Rushby, and others.

PVS has a rich **specification language** to define theories.

PVS has many powerful **inference rules** to prove theorems **interactively**.

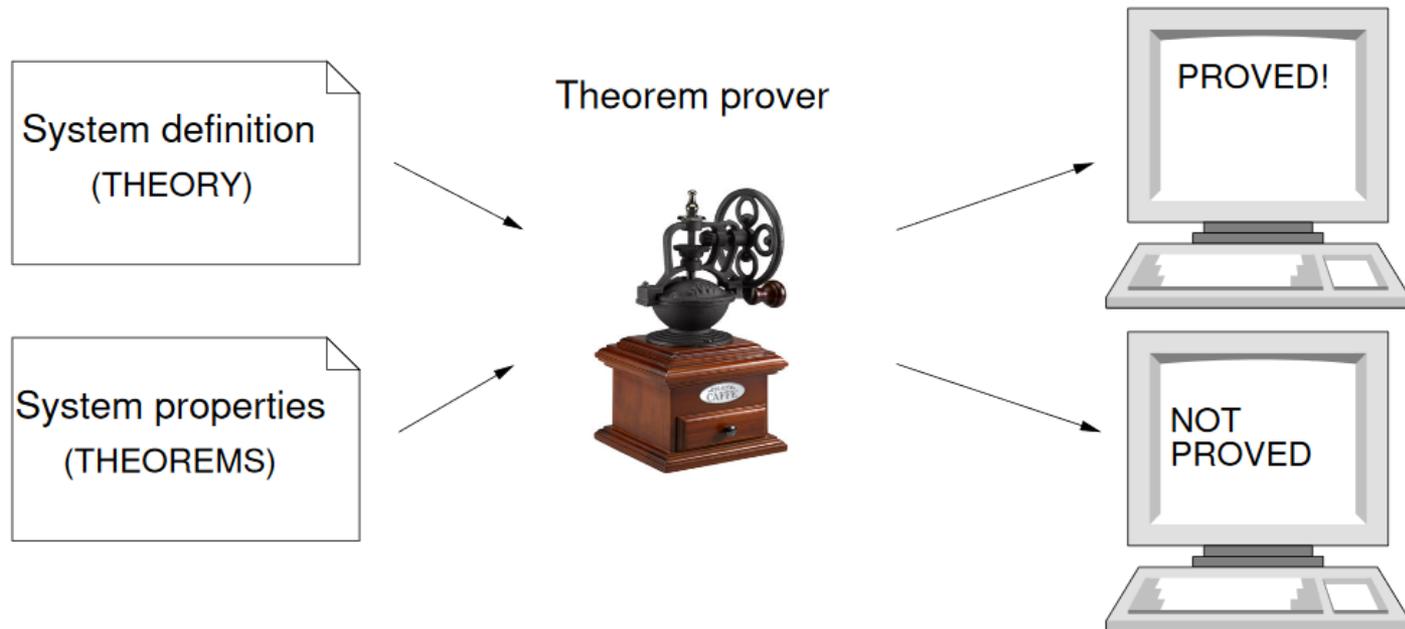
- A user submits a theorem, then chooses inference rules until the proof ends successfully, or gets stuck.
- Often a single step is sufficient.

PVS has a **simulation extension** (*PVSio*). *PVSio* generates for each function in the declarative PVS language a procedure to compute its value (Lisp). *PVSio* can also compute functions with side effects, such as producing outputs.

Formal verification

Logic specifications model a system by stating its properties in a formal language.

Logic specifications are used for the formal verification of systems, using **automatic theorem proving**.



Formal verification

Formal verification is an important complement to simulation.

E.g., suppose we want to verify that the previously shown ICP system satisfies this property:

It is always the case that module LRI is in state LRI and its clock is reset when transition AP is executed.

```
lri_ap: LEMMA
  FORALL (s0, s1: State):
    per_APout(lri(s0)) AND s1 = APout(s0) IMPLIES
      mode(lri(s1)) = LRI AND time(lri(s1)) = 0
```

A single application of the *grind* rule (multiple simplifications) is sufficient.

```
Rule? (grind)
```

```
...
```

```
Q.E.D.
```

```
Run time = 0.17 secs.
```



Formal proof of the deterministic behavior of the pacemaker, in any heart condition.

6. Conclusions

- PVSio-web is an **open-source graphical environment** for facilitating the design and evaluation of interactive (human-computer) systems
- PVSio-web has been successfully used for
 - analyzing **commercial, safety-critical medical devices** (reverse engineering from the product) to fix existing issues, and identify new potential issues in advance
 - **medical device design**, by both formal methods experts and non-technical end users
 - creating **training material** for device developers and device users, and **raising awareness** of criticalities during training sessions

Conclusions

*PVSio-web has been used for demonstration of user interface issues with medical devices in use at UCLH and in other UK hospitals in the **CHI+MED** (Computer-Human Interaction for Medical Devices, EP/G059063/1) project, funded by the *Engineering and Physical Sciences Research**



PVSio-web has been used for modelling contrast media injectors at Department of Diagnostic and Interventional Radiology of the University of Pisa, Italy, in the framework of the "Centro interdipartimentale di ricerca in Promozione della Salute e Information Technology" of Pisa (ProSIT).



US Food and Drug Administration (FDA) & The Medicines and Healthcare products Regulatory Agency (MHRA) are using these results and trialling these methods on premarket reviews

