

# **Modelagem e Implementação de Escalonadores de Tempo Real para Sistemas Embarcados**

**Hugo Marcondes<sup>1</sup>, Rafael Cancian<sup>2</sup>  
Marcelo Stemmer<sup>2</sup>, Antônio Augusto Fröhlich<sup>1</sup>**

<sup>1</sup>Laboratório de Integração de Software e Hardware  
Universidade Federal de Santa Catarina  
Caixa Postal 476 – 88049-900 – Florianópolis – SC – Brazil

<sup>2</sup>Departamento de Automação e Sistemas  
Universidade Federal de Santa Catarina

{hugo,guto}@lisha.ufsc.br,{cancian,marcelo}@das.ufsc.br

**Abstract.** *Embedded systems frequently require an integrated hardware-software design within real time constrains. In order to achieve such contrains, an adequate selection of a scheduling policy must be done. This work proposes the design and implementation of real time schedulers for embedded systems, within the context of Application Oriented System Design (AOSD). The use of AOSD enabled the development of schedulers where the policy is detached from the scheduling mechanism, fostering a better reusability of the scheduling components. The results shows that such design could be implemented to scale from 8 bits microcontrollers, 32 bits architectures and to specific hardware implemented design.*

**Resumo.** *Devido a suas características, sistemas embarcados freqüentemente demandam um projeto integrado de software e hardware com restrições de tempo real. Para que tais restrições sejam respeitadas, uma política de escalonamento de tarefas adequada deve ser selecionada. Este trabalho apresenta a modelagem e implementação de escalonadores de tempo real para sistemas embarcados, no contexto do projeto de sistemas orientados à aplicação. Esta abordagem permitiu a separação da política de escalonamento e seu mecanismo, promovendo uma maior reusabilidade dos artefatos envolvidos. Os resultados apresentados demonstram que esta implementação permite o seu uso em microcontroladores de 8 bits, arquiteturas de 32 bits, e até mesmo para implementações dedicadas de hardware.*

## **1. Introdução**

Sistemas operacionais para sistemas dedicados devem ser adaptados para prover apenas o suporte necessário a uma aplicação bem definida. Desta forma, fatorar o sistema operacional em componentes selecionáveis e configuráveis é uma boa alternativa para modelar e projetar sistemas operacionais dedicados. Por outro lado, sistemas dedicados freqüentemente demandam um projeto integrado de software e hardware e apresentam uma grande variedade em termos de arquiteturas de hardware, desde microcontroladores de 8 bits até ao projeto de chips dedicados (ASIC), dificultando a tarefa de modelar e implementar

componentes reusáveis que possam ser efetivamente aplicados nessa gama de arquiteturas.

Dentre as principais famílias de componentes que formam um Sistema Operacional Embarcado (SOE) estão as tarefas, os temporizadores, os sincronizadores e os escalonadores de tarefas. Entretanto, essas famílias de componentes estão intrinsecamente relacionadas, de modo que, sem o devido projeto, um componente não pode ser modificado/adaptado sem influenciar os demais. No caso de sistemas de tempo real, tais componentes devem ainda ser projetados de forma a garantir restrições tais como a previsibilidade do tempo máximo de execução destes.

Mesmo numa única família, a adaptação de componentes para diferentes cenários de execução pode não ser fácil. Escalonadores de tarefas, por exemplo, apresentam uma miríade de algoritmos e características, incluindo escalonadores de tempo real altamente especializados e relativamente complexos. Nesses casos, permitir que um sistema operacional embarcado possua suporte a qualquer algoritmo (tempo real ou não), independente de suas características, sem exigir alterações no código de outras partes e componentes do sistema, é um grande desafio; principalmente em um sistema embarcado onde há grandes restrições de recursos computacionais.

Além das grandes diferenças algorítmicas e conceituais nos escalonadores, o projeto integrado de software e hardware desses sistemas permite que as funcionalidades tipicamente encontradas nos sistemas operacionais possam ser implementadas em hardware, seja através de dispositivos de lógica programável e até mesmo através do projeto de ASICs, sendo comum a implementação de escalonadores de tarefas em hardware, dado que esse é um componente executado com muita freqüência e fonte de sobrecusto (*overhead*). Assim, um sistema operacional embarcado adaptável à aplicação deve permitir que esse componente seja implementado em ambos os domínios (software e hardware) de forma eficiente.

Todos esses problemas não podem ser resolvidos simplesmente com uma implementação cuidadosa, e demandam um projeto de sistema adequado e engenhoso. Neste artigo, focamos na descrição da análise e modelagem dos principais componentes relacionados aos escalonadores e aspectos avançados de implementação que, apenas juntos, permitem a solução adequada dos problemas apresentados. Para a modelagem foi utilizada a metodologia de engenharia de domínio, segundo a AOSD (*Application Oriented System Design*) [Fröhlich 2001], que agrupa uma série de paradigmas de programação para guiar o projeto de sistemas adaptados à aplicação. Nossas contribuições incluem um modelo eficiente e a apresentação de técnicas de implementação para adaptação de escalonadores em sistemas operacionais embarcados orientados à aplicação, além de permitir sua execução em diferentes arquiteturas, incluindo pequenos microcontroladores de 8 bits.

Este artigo está organizado da seguinte forma: A seção 2 apresenta conceitos e a descrição de alguns escalonadores encontrados na bibliografia, bem como a descrição das principais técnicas utilizadas neste trabalho. As seções 3 e 4 apresentam o desenvolvimento realizado e resultados alcançados, o que inclui o modelo conceitual básico e aspectos da implementação realizada. Por fim, a seção 5 apresenta algumas conclusões e considerações finais.

## 2. Fundamentação e Trabalhos Relacionados

O escalonamento de tarefas é considerado o coração de um sistema operacional, e dezenas de diferentes escalonadores têm sido propostos, principalmente escalonadores de tempo real para classes de aplicações específicas. Muitos podem ser implementados através da ordenação em uma fila de tarefas prontas, conforme um critério determinado. Isso inclui os escalonadores mais conhecidos, como FIFO, alternância circular (*round-robin*), prioridade, SPF (*Shortest Process First*), RM (*Rate Monotonic*), EDF (*Earliest Deadline First*) entre vários outros [Harvey DEITEL 2005]. Entretanto, outros algoritmos são muito mais complexos. Algoritmos como DSS (*dynamic sporadic server*) e *dynamic priority exchange server* [Buttazzo 1997] permitem o uso de filas separadas para tarefas periódicas e aperiódicas e também que pelo menos uma tarefa periódica especial atenda tarefas aperiódicas com regras específicas de temporização, consumo e concessão de “créditos” (*budgets*); Algoritmos como *Elastic Task Model* [G.C. Buttazzo and Abeni 1998] permitem mudar parâmetros das tarefas, como seu período, para adaptar-se à carga atual do sistema. Vários outros exemplos poderiam ilustrar as grandes diferenças entre as dezenas de escalonadores de tarefas propostos na literatura, de modo que suportá-los de forma transparente não é uma tarefa óbvia.

Por sua importância, vários sistemas operacionais embarcados já permitem a adaptação de seus escalonadores, mesmo dinamicamente. Entretanto, essa adaptação normalmente restringe-se a alguns poucos algoritmos específicos que alteram apenas a ordenação de uma fila, como FIFO, *round-robin*, prioridade, EDF e RM. Algumas poucas soluções permitem a substituição por algoritmos mais elaborados. Entre elas está o S.Ha.R.K. (uma evolução do Hartik) [Shark 2008], que inclui algoritmos como PS (*Polling Server*), DS (*Deferrable Server*), SS (*Sporadic Server*), CBS (*Constant Bandwidth Server*) e CBS-FT (*Fault Tolerant CB*). Além disso, muitos sistemas operacionais de tempo real usam algoritmos de escalonamento que não consideram o prazo máximo de execução das tarefas (*deadline*), ou seja, usam escalonadores não tempo real para escalar tarefas tempo real. Entretanto, além de exigir garantias de atendimento dos prazos das tarefas, muitas aplicações embarcadas de tempo real exigem ainda mais dos escalonadores. Dentro vários exemplos, podemos citar aplicações multimídia, que exigem que a taxa de execução das tarefas de vídeo e áudio seja constante (minimizando o *jitter*). Para atender esse tipo de exigência faz-se necessária a utilização de algoritmos específicos, como o CBS [G.C. Buttazzo and Abeni 1998], já que os algoritmos usuais não são adequados, pois desconsideram totalmente o *jitter*. Disso conclui-se que não há um suporte adequado a esse tipo de aplicação quando o SOE não provê escalonadores específicos, e esse é o caso de muitos SOE, inclusive de tempo real.

Grande parte dos sistemas operacionais de tempo real disponíveis hoje, tal como o *Embedded RT Linux*, QNX e VxWorks, tem seu uso prático em plataformas profundamente embarcadas limitado, devido ao tamanho do código gerado e a dificuldade de portabilidade. Além disso, a grande maioria dos sistemas operacionais de tempo real não consideram co-design em seu projeto e, portanto, ignoram as possibilidades de configuração do hardware. Assim, embora haja muitas alternativas supostamente disponíveis de sistemas operacionais de tempo real para sistemas embarcados, poucas realmente são aplicáveis a várias arquiteturas (principalmente aquelas com maiores restrições de recursos) e adaptáveis às necessidades das aplicações-alvo.

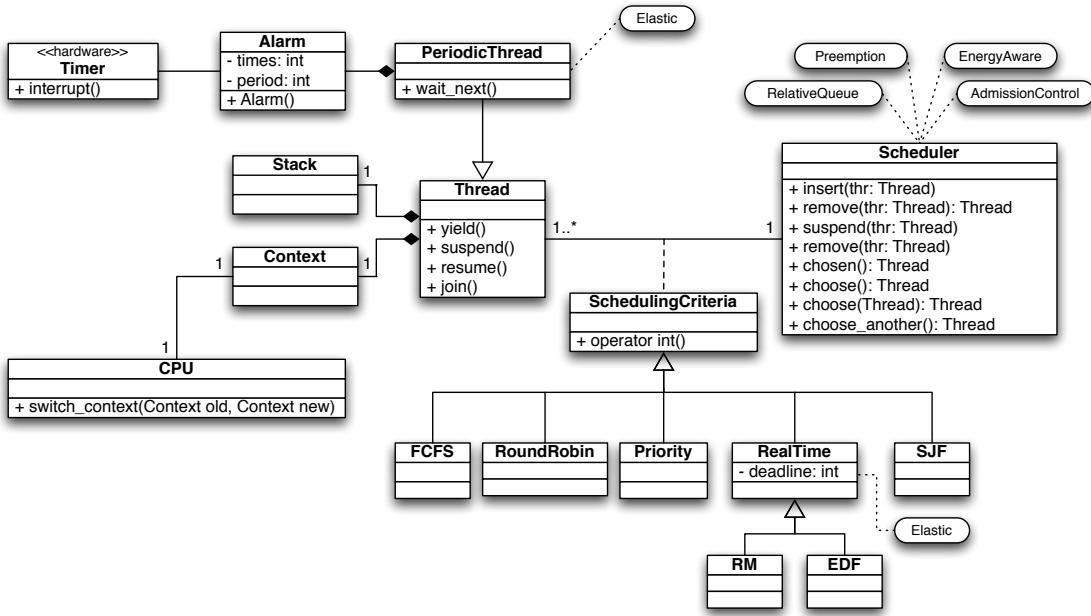
Vários trabalhos de hardware/software co-design para sistemas de tempo real já foram propostos. Implementações em hardware para escalonadores de tarefas foram propostos, dentre outros, por [Mooney and Micheli 2000], que implementaram um escalonador cíclico, e por [P. Kuacharoen and Mooney 2003], que implementaram os algoritmos de prioridade, RM e EDF. Além do suporte para escalonamento de tarefas, [Kohout and Jacob 2003] desenvolveram suporte para gerenciamento do tempo e de eventos, pois são atividades muito comuns aos STR e com alto paralelismo intrínseco. Entretanto, uma limitação desse suporte é que apenas escalonamentos com prioridade fixa são possíveis. O projeto *HThread* [Anderson et al. 2006] propõem um modelo de programação que permite que tarefas implementadas em hardware interajam com tarefas em software, através da implementação de escalonadores e dispositivos de sincronização em ambos os domínios (hardware e software). Outros tipos de suporte também foram propostos, como o de gerenciamento de memória [Shalan and Mooney 2000] e o de protocolos de acesso a recurso [Akgul 2003], que implementa o protocolo de herança de prioridade para evitar *deadlocks* e o bloqueio ilimitado de tarefas.

Nesse cenário, o EPOS (*Embedded Parallel Operating System*) surge como uma alternativa viável de sistema operacional de tempo real multiplataforma para sistemas embarcados. O EPOS comprehende um *framework* e ferramentas para geração de sistemas operacionais, sendo um produto da *Application-Oriented System Design* (AOSD) [Fröhlich 2001], que combina diversos paradigmas de projeto que visam guiar o desenvolvimento de componentes de software altamente adaptáveis e reutilizáveis. A AOSD incluiu avanços como os adaptadores de cenários [Fröhlich and Schröeder-Preikschat 2000] e os mediadores de hardware [Polpeta and Fröhlich 2004] que permitem grande eficiência na geração automática de sistemas operacionais dedicados à aplicação. Posteriormente o EPOS foi expandido para gerar automaticamente não apenas o suporte de software, mas também o suporte de hardware (IPs - *Intellectual Properties*) necessários e suficientes para a aplicação, ou seja, a geração automática de SoCs (*Systems-on-a-chip*) orientadas à aplicação, já detalhado em publicações anteriores [Polpeta and Fröhlich 2005]. Atualmente EPOS tem portes funcionais para diversas arquiteturas entre elas, IA32, PPC, SparcV8, MIPS e AVR.

### **3. Análise e Modelagem**

O processo de análise e modelagem de escalonadores de tarefas tempo real iniciou com a realização da engenharia deste domínio, de acordo com a AOSD, de forma a identificar as principais semelhanças e diferenças entre os conceitos do domínio, viabilizando desta forma a identificação das entidades que compõem o domínio de escalonamento de tempo real. A figura 1 apresenta o modelo conceitual de classes destas entidades.

Neste modelo conceitual, as tarefas são representadas através da classe *Thread*, que define o fluxo de execução da tarefa, implementando as funcionalidades tradicionais deste tipo de abstração encontrada na literatura. Esta classe também atende apenas aos requisitos de tarefas aperiódicas. A definição de uma tarefa periódica é realizada através de uma especialização da classe *Thread* que agrupa a esta mecanismos para a reexecução do fluxo de forma periódica, através do uso da abstração *Alarm*, responsável por reativar a tarefa sempre que um novo período se inicia. A classe *Alarm* por sua vez utiliza um *Timer* que irá fornecer o gerenciamento da passagem do tempo para o *Alarm*. Cada *PeriodicThread* possui seu próprio *Alarm* (embora alarmes possam



**Figura 1. Modelo proposto para escalonadores de tarefas**

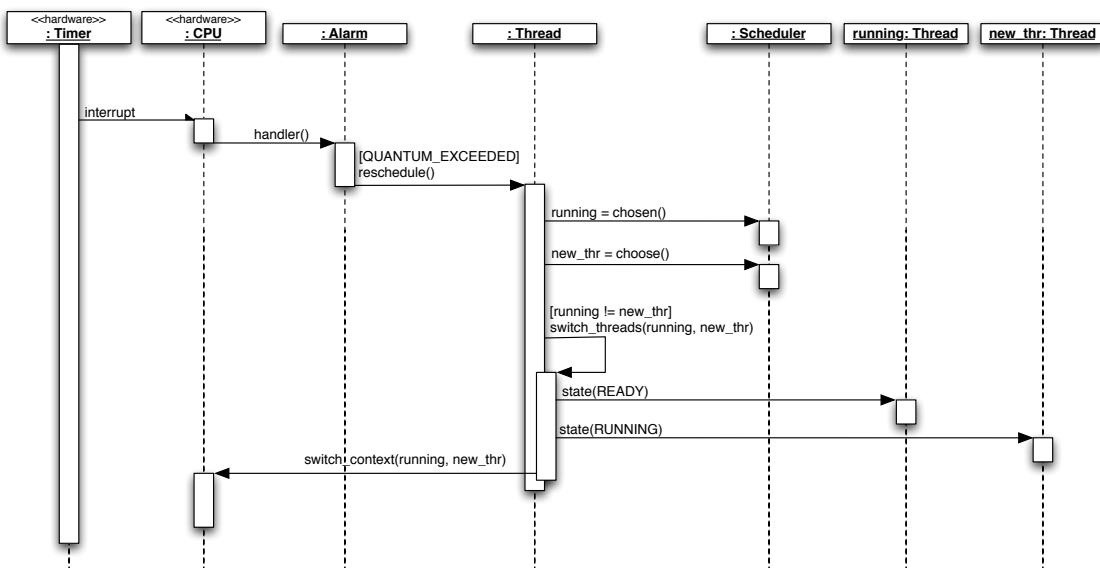
existir sem threads periódicas), que podem compartilhar o(s) *timer(s)* da arquitetura. Cabe destacar que este é um modelo conceitual, e não um modelo de implementação. Assim, por exemplo, embora um *Alarm* conceitualmente utilize um *Timer* de hardware, essa associação exige a implementação de um mediador de hardware (*driver*) do *Timer*, que não aparece no modelo. Outras classes relacionadas também não foram apresentadas pois não estão diretamente associadas ao escalonador, foco deste trabalho.

As classes *Scheduler* e *SchedulingCriteria* definem a estrutura para realizar o escalonamento das tarefas. Ressalta-se umas das principais diferenças entre as abordagens tradicionais de modelagem de escalonadores, que geralmente apresentam uma hierarquia de especializações de um escalonador genérico, de forma a extende-lo para outras políticas de escalonamento. De forma a reduzir a complexidade de manutenção do código, geralmente ocasionada pelo uso de uma hierarquia complexa de especializações, assim como promover o reuso de código, o modelo separa a sua política de escalonamento (*scheduling criteria*) de seu mecanismo (implementação de filas). Esta separação é uma decorrência do processo de engenharia de domínio que permitiu identificar os aspectos comuns a todas as políticas de escalonamento, permitindo a separação destes aspectos (contidos no componente *Scheduler*) da caracterização de tais políticas (suas diferenças, expressas no componente *SchedulingCriteria*).

Esta separação entre o mecanismo e a política de escalonamento foi fundamental para a implementação de escalonadores em hardware. De fato, o escalonador em hardware implementa apenas o mecanismo de escalonamento, que realiza a ordenação das tarefas baseado na política selecionada. Isto permite que o mesmo componente em hardware possa ser utilizado independente da política selecionada. A separação do mecanismo de escalonamento e a sua política é realizada através da separação do algoritmo de ordenamento e o mecanismo de comparação entre os elementos que compõem a lista

do escalonador. Desta forma, cada política de escalonamento define a maneira como os elementos serão ordenados na respectiva fila.

Adicionalmente, durante o processo de análise e de engenharia de domínio foi identificada uma série de características que, segundo a AOSD, definem propriedades configuráveis (configurable features) de seus componentes [Fröhlich 2001]. De fato, tais características representam pequenas variações de uma entidade do domínio que podem ser ativadas ou não, alterando de forma sutil o comportamento do mesmo. Dentre tais propriedades configuráveis, foi identificada a característica do escalonamento ser pre-emptivo, o controle de admissão das tarefas, assim como a consideração de parâmetros de consumo de energia, permitindo que o mecanismo realize também políticas de qualidade de serviço (QoS) [Wiedenhoft and Fröhlich 2008].



**Figura 2. Diagrama de seqüência do reescalonamento de tarefas.**

Outra característica identificada é relativa aos escalonadores que precisam alterar propriedades do modelo de tarefas utilizado. Conforme visto na seção de fundamentação, algoritmos de escalonamento elástico (como o *elastic task model*), permitem que o período das tarefas periódicas possam ser aumentados, caso a taxa de utilização da CPU esteja alta, e depois restaurados. Outros escalonadores não triviais, como o CBS e DSS possuem característica análoga. Essa característica é modelada como uma propriedade configurável aplicável aos `SchedulingCriteria` relativos a tarefas periódicas, assim como a classe `PeriodicThread`, habilitando funções que alteram a periodicidade ou outra propriedade das tarefas, uma vez que a política de escalonamento a solicite. Desta forma, mesmo algoritmos complexos podem ser suportados e adaptados sem exigir especializações de classes que complicariam o projeto.

De forma a ilustrar as interações entre os componentes do escalonador proposto, a figura 2 apresenta a interação dos componentes durante o rescalonamento ocorrido quando a fatia de tempo concedida para a tarefa em execução termina. Neste contexto, o Timer é responsável por gerar interrupções periódicas, que são contadas pelo Alarm. Quando o estouro da fatia de tempo concedido a tarefa atual é expirado, o Alarm invoca o método

da classe Thread para solicitar o reescalonamento das tarefas. Este por sua vez irá verificar qual é a tarefa que está em atual execução, assim como invocar o método *choose()* do Scheduler. Este retorna um ponteiro para a tarefa que deve ser executada. Neste ponto é realizada uma verificação para identificar se é necessário realizar uma troca de contexto de execução para uma tarefa de maior prioridade. Caso a troca seja necessária, os estados das tarefas envolvidas no processo são atualizados e uma troca de contexto é realizada na CPU; caso contrário o processo finaliza, mantendo a tarefa atual em execução.

Novamente, ressalta-se que a figura 2 representa um modelo conceitual, e não de implementação. As classes Timer e CPU apresentadas representam hardware, embora possuam implementação de seus respectivos mediadores em software. As mensagens interrupt e handler, deste modo, não representam métodos implementados em software, mas sim o sinal elétrico de interrupção e a invocação, pelo hardware, do tratador dessa interrupção, respectivamente. A mensagem switch-context, por sua vez, corresponde a um método implementado no mediador de hardware da CPU.

## 4. Implementação e Resultados

Esta seção apresenta os detalhes de implementação dos principais componentes do escalonador proposto, em especial a implementação do mecanismo de escalonamento em software e em hardware. Em seguida é apresentado como as principais políticas de escalonamento baseado foram implementadas através das SchedulingCriteria.

### 4.1. Escalonador em software

A implementação do escalonador em software segue o modelo tradicional de lista. Esta lista pode ser configurada para ser implementada utilizando uma ordenação convencional de seus elementos, assim como uma ordenação de forma relativa, onde cada elemento armazena o seu parâmetro de ordenamento pela diferença deste com o elemento anterior, ou seja, seu parâmetro será sempre relativo a seu predecessor e assim por diante. Esta estrutura se torna especialmente interessante na implementação de políticas que possuem o seu ordenamento influenciado pela passagem do tempo, como o algoritmo EDF. Uma vez que o tempo é sempre crescente (como o *deadline* absoluto, critério utilizado pelo EDF), a utilização de uma lista convencional resultará em um estouro do limite das variáveis após um certo tempo. Esse tempo pode ser de apenas algumas horas, dependendo da freqüência, em um microcontrolador de 8 bits, causando o comportamento incorreto e inesperado do escalonador, o que é inadmissível em um sistema de tempo real. O uso de uma lista relativa nesses casos, elimina o problema de estouro de variável, cuja ocorrência é dificilmente detectada.

Independente do uso de filas relativas ou convencionais, o critério utilizado pelo algoritmo de ordenamento da fila é realizado pelo SchedulingCriteria. De forma geral, este componente pode ser visualizado como uma especialização do tipo inteiro que define o ordenamento da fila, e implementa a sobrecarga de seus operadores aritméticos e de comparação, de forma a estabelecer políticas mais complexas de ordenamento, exigidas por vários algoritmos. Por exemplo, no caso de algoritmos multifilas, a SchedulingCriteria pode encapsular dois parâmetros de ordenamento: a identificação da fila e a prioridade do elemento dentro desta fila, além de sobrepor o operador de comparação menor/igual ( $\leq$ ) para que ambos os parâmetros sejam avaliados durante a comparação entre dois elementos, durante a sua inserção ordenada no

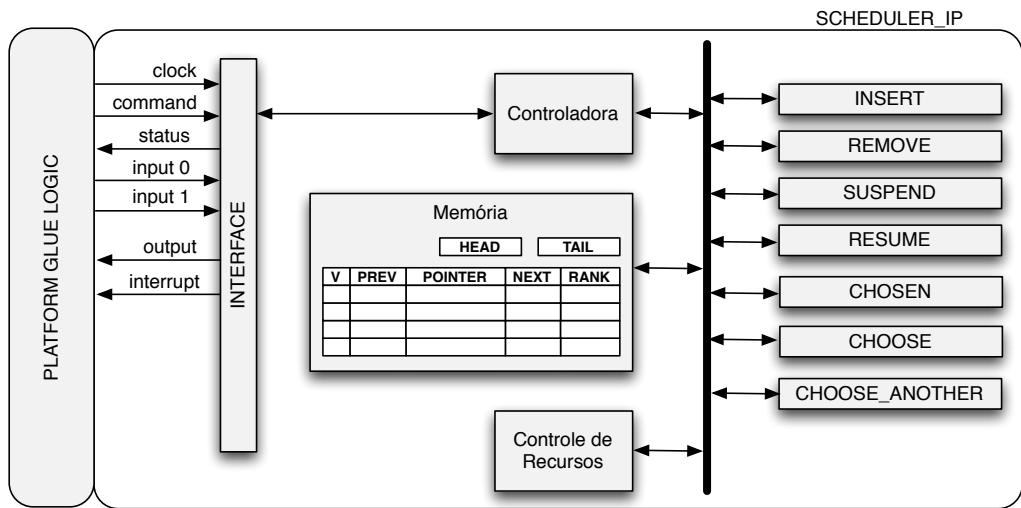
mecanismo de escalonamento. O uso de sobrecarga de operadores mantém o projeto elegante, evita a grande hierarquia de classes e provê suporte adequado a algoritmos mais complexos.

A figura 3(a) apresenta alguns trechos da implementação dos critérios de escalonamento, em que pode-se observar a sobrecarga do operador `int()` (linha 6) e do operador  `$\leq$`  (linha 14). A figura 3(b) apresenta parte do escalonador metaprogramado, tornando-se independente da entidade `T` a ser escalonada (normalmente `Thread`) e da implementação da lista (também metaprogramada). A figura 3(c) apresenta parte do arquivo de configuração do SOE em que é possível especificar a política e o mecanismos a serem utilizados.

```

1  namespace Scheduling.Criteria {
2
3    class Priority {      // Priority ( static and dynamic)
4      public:
5        Priority (int p = NORMAL): _priority(p) {}
6        operator const volatile int () const volatile { return _priority ; }
7        void priority (int p) { _priority = p; }
8      protected: volatile int _priority ;
9    };
10
11    class RealTime: public Priority {
12      public:
13        RealTime(int p, const RTC::Microsecond & deadline): Priority (p),
14          _relDeadline (deadline) {} // Aperiodic real-time
15        bool operator <=(const RealTime & other) const {return (int)*this
16          < (int)other; }
17
18      private: RTC::Microsecond _relDeadline;
19
20    class Periodic : public RealTime { ... };
21
22    class EDF: public Periodic {
23      public:
24        EDF(int p): Periodic (p), _activations (0), _phase(0) {}
25        EDF(const RTC::Microsecond & deadline):Periodic (deadline,
26          deadline, INFINITE,0), _phase(0), _activations (0) {}
27
28      private:
29        RTC::Microsecond _phase;
30        unsigned int _activations ;
31    };
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
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
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
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
745
746
747
747
748
749
749
750
751
752
753
754
755
756
756
757
758
758
759
759
760
761
762
763
764
765
765
766
767
767
768
768
769
769
770
771
772
773
774
775
775
776
777
777
778
778
779
779
780
781
782
783
784
784
785
785
786
786
787
787
788
788
789
789
790
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560

```



**Figura 4. Diagrama de blocos do componente escalonador em hardware.**

É importante ressaltar dois aspectos da implementação deste componente devido a restrições inerentes de sua implementação em hardware, em específico com o foco em dispositivos de lógica programável.

Cabe destacar dois aspectos da implementação deste componente, que foram considerados devido às restrições existentes em dispositivos de lógica programável, os quais foram utilizados para prototipação deste componente. Tais aspectos estão relacionados com a restrição de recursos provenientes destes dispositivos. Idealmente um escalonador em hardware deveria explorar ao máximo o paralelismo inerente deste, contudo, o custo pela exploração máxima deste paralelismo em termos de consumo de blocos lógicos da FPGA se torna extremamente alto, principalmente na implementação da comparadores paralelos para realizar a busca de elementos e também a procura pela posição de inserção de elementos.

Em especial, o uso de ponteiros de 32 bits, para expressar referências aos elementos armazenados na lista (neste caso Threads) torna-se muito custoso, quando se deseja realizar uma pesquisa por esses ponteiros, e consequentemente utilizar comparadores de 32-bits. Por outro lado, o número máximo de Threads a serem escalonadas em um sistema embarcado é conhecido de antemão, e por isso, foi adotado um mapeamento entre o endereçamento de objetos da arquitetura (do tamanho da palavra da arquitetura) para um endereçamento interno a este componente que irá utilizar tantos bits quanto forem necessários para endereçar apenas o número máximo de elementos que este componente suporta, reduzindo assim, a lógica gasta pela implementação do mesmo.

O outro aspecto está relacionado à busca pela posição do elemento durante a inserção na fila. Idealmente, a busca por esta posição poderia ser implementada através de uma comparação paralela de todos os elementos da lista, de forma a encontrar a posição de inserção em apenas um ciclo de execução. Contudo, esta abordagem, além de aumentar o consumo de recursos e lógica conforme já mencionado, ainda gera um impacto no atraso máximo do circuito devido à sua maior complexidade, reduzindo também a freqüência de operação do mesmo. Desta forma, se optou pela implementação de uma

busca seqüencial pela posição de inserção na lista, percorrendo a mesma. Nesta abordagem, apesar do tempo de inserção de elementos variar, essa variação pode ser, na maioria dos casos, escondida pelo paralelismo entre a execução do componente em hardware e a CPU. Desta forma, durante a operação de inserção, o controle é imediatamente devolvido a CPU, enquanto o hardware executa a inserção do elemento na fila.

#### 4.3. Avaliação da implementação do modelo proposto

A avaliação da implementação do modelo proposto foi realizada através de uma aplicação sintética de tempo real, onde foi definido um conjunto de tarefas periódicas hipotéticas. A configuração dos componentes do EPOS foi realizada através das ferramentas desenvolvidas para o sistema que gera um conjunto de parâmetros que efetuam a ligação da interface utilizada pela aplicação à respectiva implementação do componente selecionado [Figura 3(c)]. A figura 5 ilustra a implementação desta aplicação teste. A figura 5 (a) apresenta a implementação de cada tarefa, sendo que neste caso, a tarefa consome ciclos de CPU, simulando sua ocupação da CPU. A figura 5 (b) apresenta a aplicação principal, que é responsável por ativar e criar no sistema as tarefas que serão executadas (linhas 5–7), definindo os parâmetros de de cada tarefa, de acordo com a política de escalonamento utilizada, neste caso, é utilizado a política EDF, definindo desta forma o período, deadline e número de ativações que a tarefa deve ter.

<pre> 1 int Tn(int n, RTC::Microsecond wcet) { 2     RTC::Microsecond now, milieexec; 3     int activations = 0; 4 5     while (1) { 6         now = Alarm::elapsed(); 7 8         // waste time in CPU 9         milieexec = 0; 10        do { 11            while (now == Alarm::elapsed()); 12            milieexec++; 13            now = Alarm::elapsed(); 14        } while (milieexec &lt; wcet); 15 16        activations++; 17        PeriodicThread::wait.next(); 18    } 19    return 0; 20 }</pre>	<pre> 1 int main() { 2     cout &lt;&lt; "Main will create the periodic threads ...\\n"; 3 4     PeriodicThread *t1, *t2, *t3; 5     t3 = new PeriodicThread(&amp;Tn, 3, 60e3, SchedulingCriteria(300e3,300e3,10)); 6     t2 = new PeriodicThread(&amp;Tn, 2, 40e3, SchedulingCriteria(200e3,200e3,10)); 7     t1 = new PeriodicThread(&amp;Tn, 1, 20e3, SchedulingCriteria(100e3,100e3,10)); 8 9     cout &lt;&lt; "Main will wait for periodic threads to finish ...\\n"; 10    int status1 = t1-&gt;join(); 11    int status2 = t2-&gt;join(); 12    int status3 = t3-&gt;join(); 13 14    cout &lt;&lt; "Main will destroy periodic threads ...\\n"; 15    delete t1; delete t2; delete t3; 16 17    cout &lt;&lt; "Main will finish ...\\n"; 18    return 0; 19 }</pre>
(a)	(b)

**Figura 5. Aplicação de teste: (a) código de cada tarefa e (b) criação das tarefas**

Esta aplicação foi compilada para as arquiteturas PowerPC (32 bits) e AVR (8 bits), utilizando os algoritmos de escalonamento EDF, RATE MONOTONIC e PRIORIDADE. A tabela 1 apresenta o tamanho da aplicação gerada para cada política e arquitetura testada. Os testes também demonstraram o correto funcionamento das políticas utilizadas.

	PPC32		AVR8	
	.text	.data	.text	.data
EDF	51052	300	49246	853
Rate Monotonic	47908	272	36800	1003
Prioridade	47864	272	36790	1003

**Tabela 1. Memória consumida pela aplicação teste**

Os testes também foram realizados utilizando o escalonamento em hardware. Neste caso, utilizando uma plataforma de experimentação da FPGA VIRTEX4, que integra um processador PowerPC 405 e blocos de lógica programável, permitindo assim a prototipação rápida de aceleradores em hardware dedicados.

# Máx. Tarefas	Logic Usage	Slices	Máx. Freq.
2	5%	326	214.6 Mhz
4	10%	551	161.5 Mhz
8	19%	1078	138.8 Mhz
16	36%	2015	123.4 Mhz
24	51%	2833	114.6 Mhz
32	73%	3997	113.4 Mhz
48	103%	5665	82.0 Mhz

**Tabela 2. Resultados de utilização da FPGA para o componente Scheduler**

O modelo de FPGA disponível nesta plataforma de experimentação (ML 403) é o modelo XC4VFX12 que provê 5,412 slices de lógica para a implementação dos aceleradores. A tabela 2 apresenta a área consumida desta FPGA, de acordo com o número máximo de tarefas que o escalonador pode gerenciar.

## 5. Conclusões

Este artigo apresentou a modelagem para a implementação de escalonadores de tarefas tempo real, de acordo com a AOSD. O uso de técnicas como a engenharia de domínio possibilitou o isolamento das diferenças presentes nas políticas de escalonamento, possibilitando um melhor reúso dos artefatos de projeto (implementação das políticas de escalonamento e do mecanismo de escalonamento), e permitiu que a abordagem proposta seja utilizada não apenas no desenvolvimento de sistemas reais, assim como uma plataforma de experimentação de algoritmos de escalonamento de tempo real, viabilizando a execução de testes de políticas existentes, ou mesmo novas políticas em todas as arquiteturas suportadas pelo sistema EPOS, que variam desde microcontroladores de 8 bits a arquiteturas de 32 bits.

## Referências

- Akgul, B. (2003). “Hardware support for priority inheritance.” In Publishers, K. A., editor, *24th IEEE International Real-Time Systems Symposium*.
- Anderson, E., Agron, J., Peck, W., Stevens, J., Baijot, F., Komp, E., Sass, R., and Andrews, D. (2006). “Enabling a uniform programming model across the software/hardware boundary”. In *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on*, pages 89–98.
- Buttazzo, G. (1997). *Hard Real-Time Computing Systems*. Kluwer Academic Publishers.
- Fröhlich, A. A. (2001). *Application-Oriented Operating Systems*. PhD thesis, Sankt Augustin: GMD - Forschungszentrum Informationstechnik.
- Fröhlich, A. A. and Schröeder-Preikschat, W. (2000). “Scenario adapters: Efficiently adapting components.” In *Proceedings of 4th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, USA.

- G.C. Buttazzo, G. L. and Abeni, L. (1998). “Elastic task model for adaptive rate control.” In *19th IEEE Real-Time Systems Symposium*, pages 286–295, Madrid, Spain.
- Harvey DEITEL, Paul DEITEL, D. R. C. (2005). *Sistemas operacionais*. Prentice Hall.
- Kohout, P. and Jacob, B. (2003). “Hardware support for real-time operating systems.” In *Proceedings of CODES - ISSS'03*, Newport Beach, CA - USA.
- Marcondes, H. and Fröhlich, A. A. M. (2008). “On hybrid hw/sw components for embedded system design.” In *Proceedings of the 17th IFAC World Congress, 2008*, volume 17, Seoul, South Korea. IFAC.
- Mooney, V. and Micheli, G. D. (2000). “Hardware/software codesign of run-time schedulers for real-time systems.” In *Proceedings of Design Automation of Embedded Systems*, pages 89–144.
- P. Kuacharoen, M. S. and Mooney, V. (2003). “A configurable hardware scheduler for real-time systems.” In *Proceedings of International Conference on Engineering of Reconfigurable Systems and Algorithms - ERSA'03*.
- Polpeta, F. V. and Fröhlich, A. A. (2004). “Hardware mediators: a portability artifact for component-based systems.” In *Proceedings of International Conference on Embedded and Ubiquitous Computing*, volume 3207, pages 271–280.
- Polpeta, F. V. and Fröhlich, A. A. (2005). “On the automatic generation of soc-based embedded systems.” In *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation*.
- Shalan, M. and Mooney, V. (2000). “A dynamic memory management unit for embedded real-time system-on-a-chip.” In *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 180–186.
- Shark (2008). S.ha.r.k.: Soft hard real-time kernel. World Wide Web.
- Wiedenhoft, G. R. and Fröhlich, A. A. (2008). “Gerência de energia no epos utilizando técnicas da computação imprecisa.” In *Proceedings of the Fifth Brazilian Workshop on Operating Systems*, pages 34–45.