# Ferramenta de Suporte ao Projeto Automatizado de Sistemas Computacionais Embarcados

**Rafael Luiz Cancian**[1], **Marcelo Ricardo Stemmer**[1],
**Alexandre Schulter**[2], **Antônio Augusto Frölich**[2]

[1]Departamento de Automação e Sistemas (DAS)
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 - CEP 88049-900 - Florianópolis, SC, Brazil

[2]Departamento de Informática e de Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)

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

***Abstract.*** *Embedded systems are comprised of hardware and software and usually run dedicated applications in environments with highly restricted resources. These systems should match applications' requirements, with minimum support. Usually, in this domain, reuse of components, architectural transparency, low overhead, and reconfigurability are essential features. The Application-Oriented System Design (AOSD) method was created to deal with these issues, and aims at guiding the development of embedded systems that exactly match applications' requirements. This paper describes a configuration and system generation tool that is being used with EPOS (Embedded and Parallel Operating System), an OS developed using AOSD. This tool receives the application source-code as input and builds the necessary computational support for this specific application. To illustrate this process, this paper shows a case study describing the generation of an embedded system to support a simple audio decoder application.*

## 1. Resumo

Sistemas embarcados são compostos de hardware e software e normalmente executam aplicações dedicadas em ambientes altamente restritivos. No projeto de sistemas embarcados temos características altamente desejáves, como o reuso de componentes, transparência arquitetural, baixo overhead, e reconfigurabilidade. O Projeto de Sistemas Orientado a Aplicação (AOSD) foi criado para tratar dessas questões, e visa guiar o desenvolvimento de sistemas embarcados que atendam exatamentente os requisitos da aplicação. Este artigo descreve uma ferramenta para configuração e geração de sistemas embarcados que está sendo utilizada no EPOS (Embedded and Parallel Operating System), um Sistema Operacional desenvolvido usando AOSD. A partir do código-fonte da aplicação a ferramenta gera o suporte computacional necessário e sufiente à execução dessa aplicação. Para ilustrar esse processo, este artigo mostra um estudo de caso descrevendo a geração de um sistema embarcado para suportar uma aplicação simples de decodificação de áudio.

## 2. Introdução

Sistemas embarcados estão sendo usados extensivamente em vários setores da indústria para controlar praticamente qualquer tipo de equipamento. Os sistemas embarcados têm crescido muito em quantidade e também em complexidade à medida que eles se beneficiam dos avanços tecnológicos. Nesse sentido, o desenvolvimento de sistemas embarcados como Sistemas-em-um-Chip (SoC) desenvolvidos sobre Dispositivos Lógicos Programáveis (DLP) tem sido uma abordagem para tratar dessa complexidade, principalmente pois DLP permitem o desenvolvimento de sistemas complexos em pouco tempo, pois apresentam diversas vantagens, que incluem facilidade de integração dos componentes de hardware num único chip; flexibilidade e facilidade no projeto de hardware; possibilidade de efetuar o projeto conjunto de hardware e software; e do particionamento hardware/software dos componentes que compõem o sistema.

Métodos de projeto conjunto de hardware e software (hw/sw co-design) surgiram como uma solução a muitos problemas de integração entre componentes de software e hardware que ocorriam apesar dos componentes funcionarem conforme suas especificações lógicas. Nesse sentido, tem-se intensificado a quantidade de pesquisas que visam traduzir uma especificação em linguagem de programação para uma descrição de hardware sintetizável. Os principais trabalhos incluem [Ku and DeMicheli 1990], que propuseram HardwareC; [Galloway 1995] desenvolveu Transmogrifier C; e [Graphics ] desenvolveu Catapult C que sintetiza C++. Alguns outros trabalhos visam sintetizar hardware a partir de níveis de abstração mais altos, como UML [Björklund and Lilius 2002] ou mais baixos, como bytecodes ou binários [Cardoso and Neto 1999] [Stitt and Vahid 2002] [G. Mcgregor and Stitt 2005].

Com a utilização de componentes de sofware e de hardware, o particionamento hardware/software é considerado uma das etapas mais importantes, pois nessa etapa são exploradas várias alternativas de projeto (composição e mapeamento de componentes) visando encontrar uma solução ótima (ou sub-ótima) em relação a certas restrições impostas pelo projetista, principalmente em relação à área de silício e à latência. Abordagens interativas e automáticas têm sido propostas. A abordagem interativa tem sido ainda muito utilizada, pois a estimativa do desempenho das várias alternativas de projeto depende de diversos aspectos e ainda é um processo complexo. A abordagem automática visa minizar a função-custo utilizada sem intervenção do projetista. Várias pesquisas têm utilizado técnicas distintas para o particionamento automático, como Integer Linear Programming [Niemann and Marwedel 1996] [et al 2000], Simulated Annealing [D. Herrmann and Ernst 1994], Tabu Search [et al 1997], e Genetic Computing [V. Srinivasan and Vemuri 1998] [et al 2003] [P. Mudry and Tempesti 2006].

Em relação ao software embarcado, deve-se apresentar à aplicação a mesma interface de programação (API) para diferentes arquiteturas, e também os mesmos componentes, de forma que o desenvolvimento desse sistema possa ser aplicado de forma transparente. Sistemas operacionais de propósito geral e hardware de propósito geral normalmente não atendem adequadamente aplicações embarcadas. A metodologia *Application-Oriented System Design* (AOSD) pode fornecer suporte adequado a aplicações de domínio específico, como demonstrado em trabalhados anteriores [Fröhlich 2001] [Fröhlich and Schröder-Preikschat 2000] [Polpeta and Fröhlich 2004] [Danillo Santos and Fröhlich 2006]. Sistemas operacionais orientados à aplicação são

compostos apenas pelos componentes de software selecionados que são adaptados para atender seus requisitos. Fornecer à aplicação um sistema operacional específico para ela requer, além de componentes de software e hardware muito bem projetados, também uma ferramenta para auxiliar a geração do sistema, uma tarefa que inclui a identificação, seleção, configuração, adaptação e composição de tais componentes.

A metodologia *Application-Oriented System Design* (AOSD) propõe estratégias para definir componentes que representem entidades significativas em diferentes domínios. As variações nas aplicações de cada domínio são tratadas como definido no Projeto Baseado em Famílias [Parnas 1976]. Abstrações do domínio independetes do cenário de execução podem ser modeladas pela AOSD e organizadas como membros de famílias de componentes. Mesmo com uma separação criteriosa, essas abstrações podem conter dependências relacionadas ao ambiente de execução ao qual são aplicadas. Para reduzir essas dependências e aumentar a reusabilidade dos componentes, a AOSD agrega o conceito principal da Programação Orientada a Aspectos (AOP), a separação de aspectos. O uso conjunto destas e de outras técnicas de engenharia de software permite o desenvolvimento de metodologias eficientes para o projeto de sistemas embarcados. Um dos primeiros sistemas operacionais desenvolvido pelo uso da AOSD foi o EPOS (*Embedded and Parallel Operating System*), proposto por [Fröhlich 2001]. EPOS é um *framework* que pode ser usado em diferentes ambientes, provendo transparência arquitetural ao suporte a aplicações embarcadas [Fröhlich and Schröder-Preikschat 2000]. Além de componentes do sistema operacional, o EPOS foi extendido para tratar também com hardware [Polpeta and Fröhlich 2004], permitindo o uso de componentes cuja implementação possa ser mapeada em software ou hardware. Diversas características apresentadas por essa abordagem a tornam uma solução promissora aos problemas que atualmente limitam o desenvolvimento eficiente de sistemas embarcados. Atualmente, EPOS permite gerar instâncias de sistemas embarcados para diversas plataformas diferentes, incluindo IA32, PPC405, SparcV8 (Leon2), MIPS (Plasma), H8, e AVR8.

Neste artigo apresentamos uma abordagem para a geração parcialmente automatizada de sistemas embarcados baseados em componentes, contribuindo na produtividade do desenvolvimento desse tipo de sistema. Além disso, a habilidade de gerar versões otimizadas de um sistema operacional e de uma plataforma de hardware para cada aplicação é de grande valor, uma vez que resulta em ganhos de desempenho. O artigo é organizado em cinco seções, sendo que a seção 3 apresenta a estrutura da ferramenta desenvolvida, a seção 4 descreve alguns detalhes da implementação dessa ferramenta, a seção 5 apresenta um estudo de caso e, por fim, a última seção apresenta as conclusões e os trabalhos futuros.

## 3. Ferramenta de Apoio do Projeto

A utilização da AOSD no projeto de sistemas operacionais e o suporte de hardware apresenta a vantagem de permitir a configuração e geração automática. O conceito de interface inflada utilizado pela AOSD (do projeto Baseado em Famílias) permite que o suporte computacional (SO + hardware) seja gerado automaticamente a partir de um conjunto de componentes de hardware e software, pois as interfaces infladas (conjunto de métodos dos membros de uma família) servem como especificação de requisitos para o sistema a ser gerado. Nossa abordagem baseia-se na configuração estática (tempo de projeto), permitindo a geração de versões otimizadas do sistema operacional (e hardware) para cada

aplicação.

O código-fonte da aplicação deve ser escrito com base nas interfaces dos componentes de um repositório, que formam a API do EPOS. Esse código-fonte é submetido inicialmente ao `Analisador`. Esse módulo procura por referências a essas interfaces, e então cria uma especificação de requisitos composta por métodos, tipos e constantes usados pela aplicação. Se um requisito pode ser satisfeito por um único componente do repositório, o `Analisador` pode selecioná-lo automaticamente. Se mais de um componente satisfaz o requisito, o `Analisador` pode, num processo interativo, apresentar os possíveis componentes (e seus custos computacionais associados) para que projetista possa escolher o mais adequado. Num processo automático, pode simplesmente manter todas os possíveis componentes para que o mais adequado possa ser determinado posteriormente (pelo `Configurador`). A principal saída gerada pelo `Analisador` é um conjunto de dependências da aplicação.

Os componentes escolhidos com a ajuda do `Analisador` são adicionados ao `Configurador` e usados para construir (pelo menos) uma árvore de dependências do sistema, que representa uma alternativa de projeto. Arquivos com regras de composição e dependências, e adaptdores de cenários também são usados para esse fim. O `Configurador` então pode manter informações sobre dependências da aplicação em relação aos componentes do sistema, sobre dependências dos componentes entre si e sobre suas características, e sobre as regras que devem ser seguidas para criar a configuração, uma vez que alguns membros de componentes podem ser exclusivos. Além disso, caso o módulo anterior tenha usado o processo automático, várias árvores de dependências (alternativas de projeto) podem ser geradas (por algum critério de projeto de experimentos) e, através da análise de suas características e custos associados, a árvore que representa o sistema com as melhores características deve ser selecionada (por algum algoritmo de particionamento). Essa seleção também pode se dar por um processo interativo ou automático.

O último passo no processo de desenvolvimento do sistema é realizado pelo `Gerador`. Internamente, ele gera um conjunto de chaves que associam cada interface invocada pela aplicação a um componente específico existente no repositório, e ativa os aspectos de cenário (da AOP) que foram eventualmente identificados pelo `Configurador` como necessários. Em relação ao hardware, o `Gerador` produz uma lista de mediadores que foram incluídos pelo `Configurador`, especificando quais estão associados a IPs. O `Gerador` então traduz as chaves em parâmetros para um framework de componentes metaprogramados e executa a compilação do sistema. Além disso, quando a plataforma de hardware é configurável (FPGA), o `Gerador` produz o arquivo de configuração da síntese que guarda os parâmetros para os IPs e as informações necessárias para interconectar os IPs ao SoC. Esse arquivo de configuração é escrito numa linguagem de descrição de hardware e, juntamente com os IPs selecionados, são passados a uma ferramenta de síntese de terceiros, que realiza a tradução da especificação num arquivo *bitstrem* de configuração da FPGA-alvo. Num processo interativo, os resultados da geração podem ser utilizados para obter estimativas de custos de alternativas de projeto e realimentar o processo de escolha da melhor alternativa de projeto, realizado pelo `Configurador`.

## 4. Implementação da Ferramenta

A implementação de um protótipo da ferramenta descrita na seção anterior foi realizada utilizando a linguagem Java e alguns detalhes de seu estágio atual de desenvolvimento são apresentados nesta seção.

### Analisador

O `Analisador` aplica uma técnica que involve a compilação do código-fonte da aplicação, a análise dos arquivos-objeto resultantes, e a identificação de símbolos não resolvidos relacionados a métodos e constantes do `namespace 'System'`. Deste modo ele consegue identificar o uso da API do sistema operacional. As referências à API são extraídas por um script Perl e salvas num aquivo XML. Esse arquivo é usado pela ferramenta para gerar as dependências da aplicação. A figura 1 apresenta a interface do `Analisador`. Essa interface permite ao desenvolvedor especificar o arquivo da aplicação, visualizar o código (direita), e solicitar a extração dos requisitos. Os requisitos então são exibidos numa estrutura em árvore (esquerda), que mostra as famílias de componentes e seus membros que satisfazem os requisitos. A implementação realiza o processo interativo, descrito anteriormente, embora o processo automático possa ser incluído facilmente.

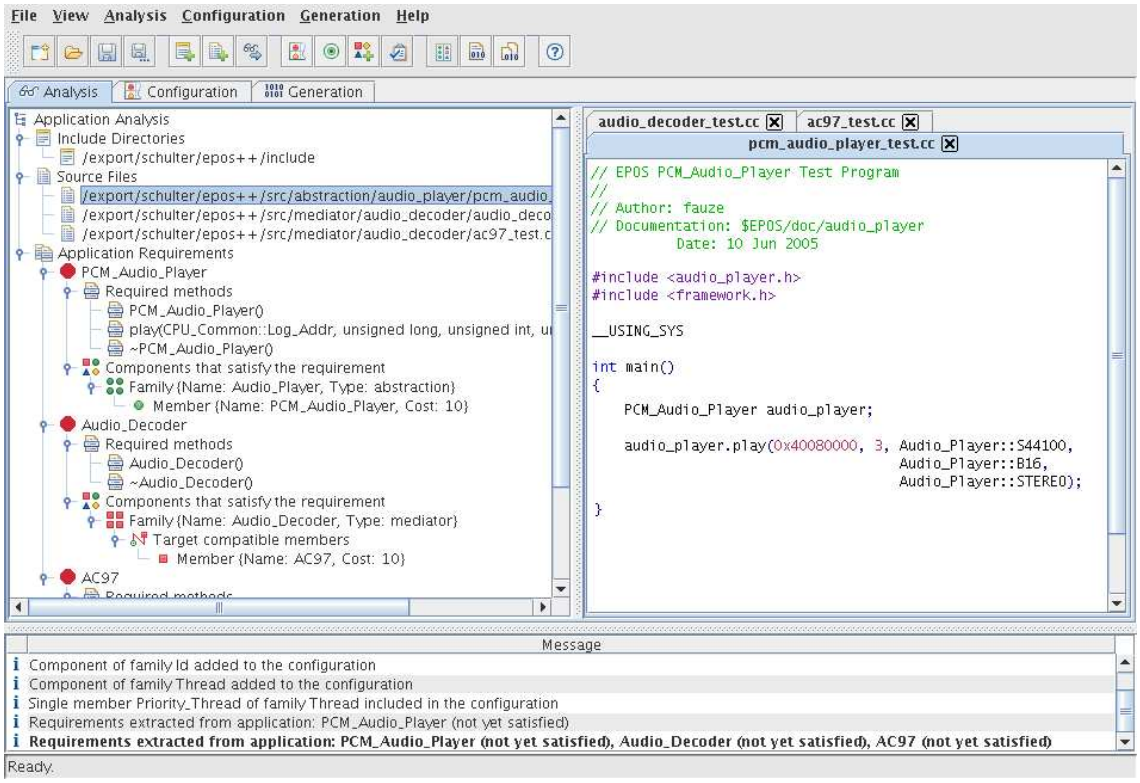


**Figure 1. A interface gráfica do `Analisador`**

### Configurador

A estratégia usada para descrever componentes no repositório e suas dependências tem papel fundamental para tornar o processo de configuração possível. A descrição dos componentes deve ser completa o suficiente para que o `Configurador` possa identificar automaticamente quais abstrações satisfazem melhor os requisitos da aplicação sem

gerar conflitos ou configurações e composições inválidas. A linguagem de descrição de componentes adotada é baseada em XML e foca na descrição individual de cada componente. Como apresentado a seguir, um componente é definido por uma família e seu conjunto de membros. A descrição do componente inclui a declaração de sua interface, um conjunto opcional de dependências e de *traits* (características configuráveis), e um pacote *common* que armazena as declarações de tipos e constantes comuns à toda família.

```
<!ELEMENT family (interface, dependency*, trait*, common, member+)>
<!ELEMENT interface (type, constant, constructor, method)*>
<!ELEMENT common (type, constant)*>
<!ELEMENT member (super, interface, trait, cost, feature,
                  dependency)*>
```

A descrição das interfaces numa família e de seus membros é a principal fonte de informação para o `Configurador`, mas a montagem de um sistema baseado em componentes vai muito além da verificação sintática de conformidade com essas interfaces. Propriedades não funcionais e comportamentais também precisam ser cobertas. Para isso, a descrição do componente inclui dois elementos especiais: `feature` e `dependency`. Esses elementos podem ser aplicados para especificar características providas pelos componentes e dependências entre componentes que não podem ser diretamente deduzidas a partir de suas interfaces. Se for uma família de mediadores de hardware ou de IPs (escritos em VHDL), as descrições dos membros também devem declarar `arch` e `mach`, que especificam a arquitetura e máquina aos quais esses membros são compatíveis. Mais detalhes e exemplos da descrição dos componentes pode ser encontrados em [Tondello and Fröhlich 2004] [Tondello and Fröhlich 2005]. Uma limitação da implementação atual do `Configurador` é gerar apenas uma árvore de dependência do sistema e, portanto, uma única alternativa de projeto. Entretanto, se os métodos associados à composição de componentes forem vinculados a uma classe (que use a mesma interface) que implemente técnicas de exploração do espaço de projeto e particionamento hardware/software, métodos automáticos podem ser incluídos sem afetar as demais etapas do processo ou o restante da implementação da ferramenta.

A interface gráfica do `Configurador` aparece na figura 2. Ela apresenta ao desenvolvedor a lista de componentes incluídos na configuração (esquerda). Através do painel de configuraçao (direita), todos os componentes, bem como a configuração da plataforma-alvo, podem ser seleciondas e modificadas com ajuda de seus respectivos *frames*. O `Configurador` inclui automaticamente os componentes necessários e, no processo interativo, permite ao desenvolvedor incluir outros (embora normalmente não seja necessário). Na implementação atual, o `Configurador` não identifca componentes desnecessários ou redundantes possivelmente incluídos pelo projetista no processo interativo.

**Gerador**

O `Gerador` permite ao projetista iniciar o processo que invoca a compilação do sistema operacional (*makefile*), causando a geração de uma instância do sistema, e o processo que invoca as ferramentas de síntese que constroem a plataforma de hardware (atualmente utilizam-se as ferramentas da Xilix - ngdbuild, map, par, trce). Nossa abordagem visa a geração de sistemas reais, e não apenas simulações. A figura 3 apresenta a interface do `Gerador`, e basicamente mostra a saída dos processos de compilação e

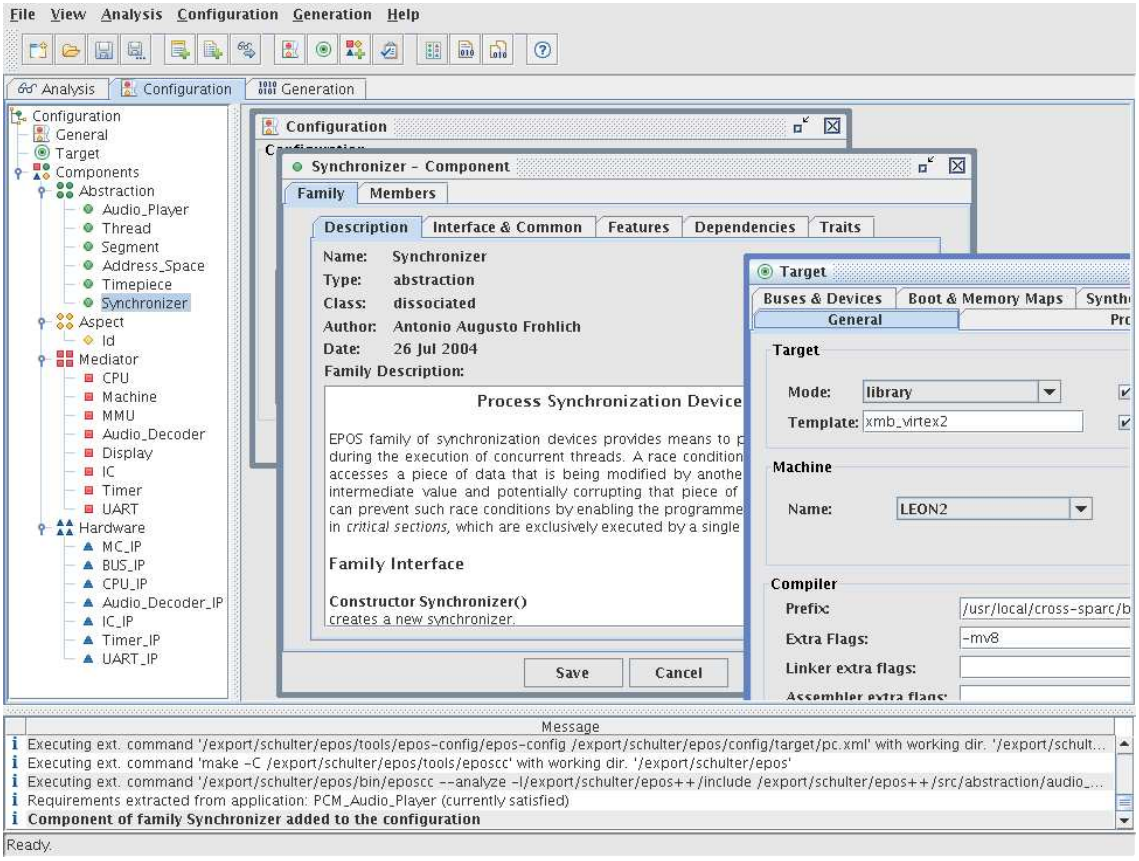


**Figure 2. A interface gráfica do `Configurador`**

síntese. Uma limitação da implementação atual é a falta de geração prévia de estimativas sobre o sistema real (área de silício, memória, latência, consumo de energia, etc) antes de realmente gerá-lo. Entretanto, se os métodos associados à geração forem vinculados a uma classe que implemente técnicas de simulação (possivelmente em vários níveis) e obtenção de estimativas, métodos automáticos podem ser incluídos sem afetar as demais etapas do processo ou o restante da implementação da ferramenta.

## 5. Estudo de Caso e Resultados

De modo a avaliar a implementação do protótipo da ferramenta descrita e seu suporte ao projeto de sistemas embarcados, foram realizados alguns estudos de caso envolvendo a geração semi-automática (processo interativo) de diferentes tipos de aplicações embarcadas para diferentes arquiteturas, incluindo AVR, SparcV8, PPC405 e IA32. Neste artigo apresentamos uma simples aplicação de decodificação de áudio para ilustrar o processo. Esse estudo de caso foi desenvolvido a partir do sistema operacional EPOS e uma placa de prototipação da empresa Xilinx como plataforma de hardware. Para evitar a apresentação de figuras repetidas ou similares, todas as telas apresentadas anteriormente neste artigo referem-se a este estudo de caso.

A placa da Xilinx usada como plataforma contém uma FPGA Virtex-II e, apesar de haver dois processadores PowerPC (*hard-coded*) internos à FPGA, eles não foram usados. Um processador *soft-core* Leon2 foi sintetizado com outros IPs para compor um SoC. O código-fonte da aplicação é listado a seguir. Ape-

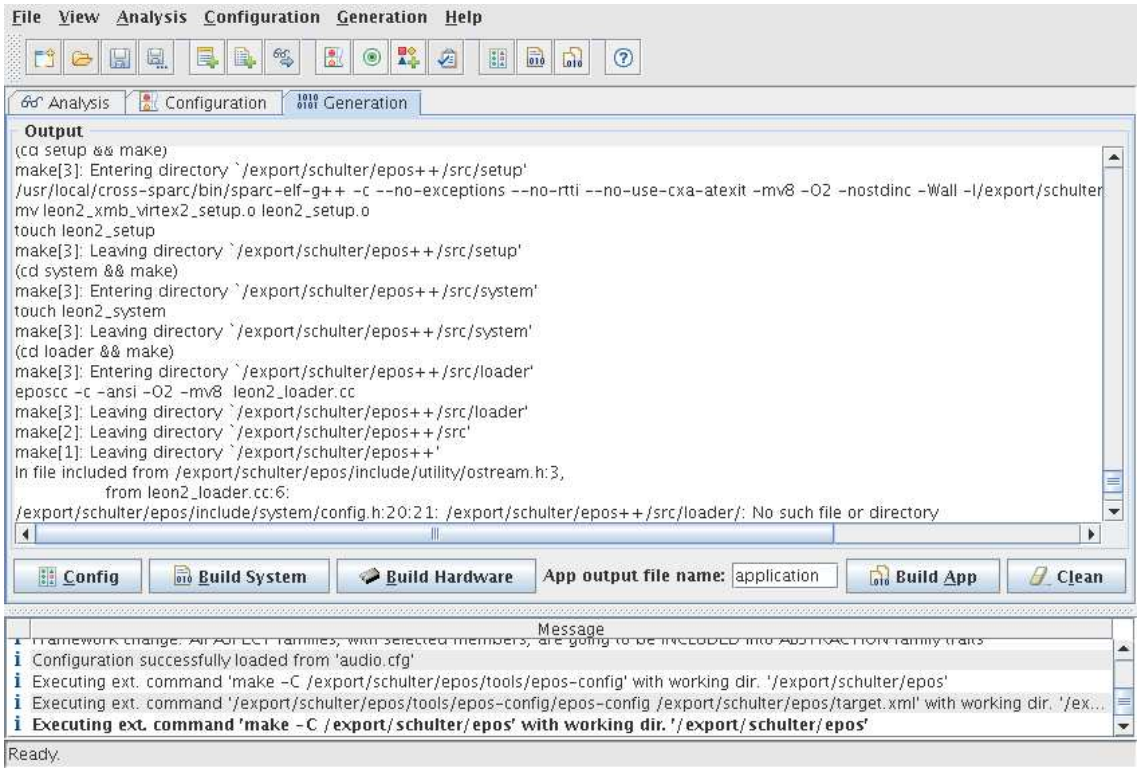


**Figure 3. A interface gráfica do `Gerador`**

nas um método da API é invocada explicitamente: o método `play(Log_Addr sound, Seconds length, Sample_Rate sample_rate, Bit_Depth bit_depth, Sound_Mode mode)` da abstração `PCM_Audio_Player`. Os parâmetros indicam que um som `16-bit stereo` amostrado à taxa de `44.1 KHz` e armazenado no endereço `0x40080000` deve ser executado por `3` segundos. A abstração `PCM_Audio_Player` fornece serviços relacionados ao controle e reprodução de áudio não compactado no formato PCM (*Pulse Code Modulation*), que é um padrão para áudio digital em computadores e CDs. A figura 4 apresenta os componentes incluídos na configuração deste estudo de caso e as dependências ente eles. O sistema final gerado pela ferramenta incluiu unicamente os componentes necessários e executou adequadamente na plataforma física com a Virtex-II.

Várias outras aplicações, componentes, e plataformas-alvo foram testados com a ferramenta. De modo geral, é nítido o ganho de tempo de projeto ao utilizar a ferramenta desenvolvida. Uma vez conhecida a API do EPOS, o projetista concentra-se exclusivamente na programação da aplicação, ignorando detalhes do suporte de software e da arquitetura. Uma vez concluída a aplicação, seu código é submetido à ferramente que, em poucos segundos ou minutos (dependendo da interação com o projetista), gera uma instância eficiente e funcional capaz de executar na plataforma selecionda. Seus pontos fortes incluem uma operação rápida e simples, e seu projeto interno garante fácil manutenção e expansão. Outra questão a considerar é a possibilidade de usar tal abordagem e ferramenta para configurar e gerar outros sistemas operacionais baseados em componentes. Isso pode ser possível se tais sistemas possuirem algumas caracteríticas do EPOS herdadas da AOSD.

```
// PCM_Audio_Player Test Program

#include <utility/ostream.h>
#include <audio_player.h>
#include <framework.h>

__USING_SYS

int main()
{
    PCM_Audio_Player audio_player;
    audio_player.play(0x40080000, 3, Audio_Player::S44100,
                      Audio_Player::B16, Audio_Player::STEREO);
    return 0;
}
```

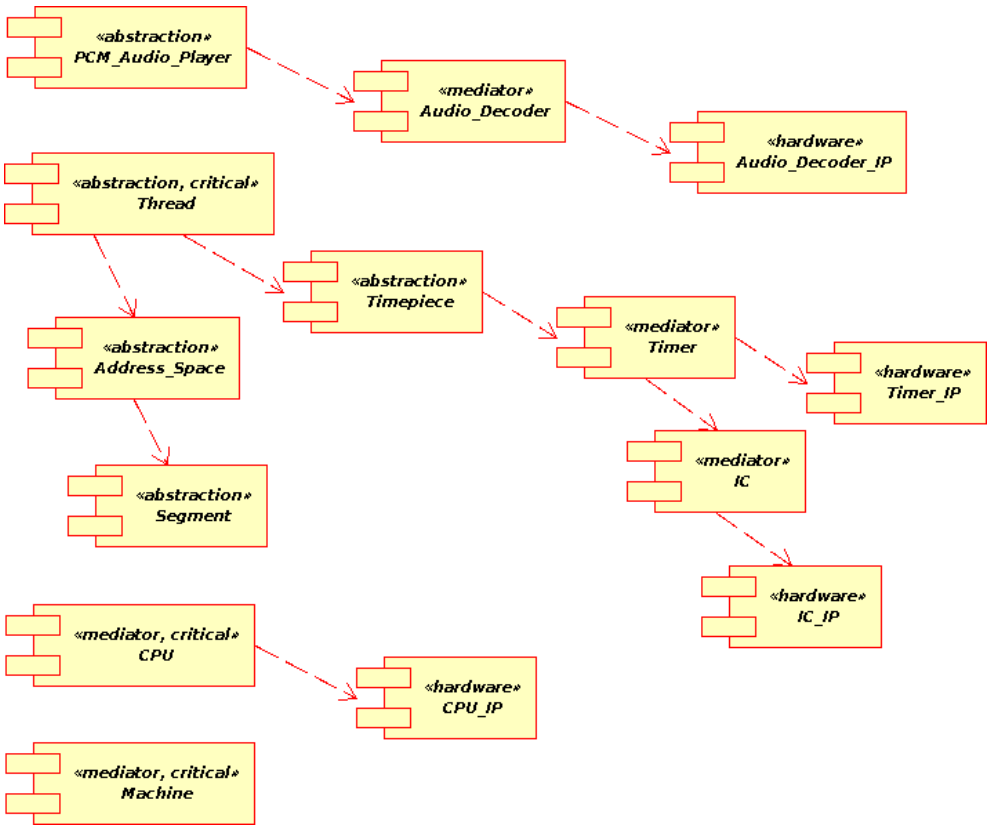


**Figure 4. Diagrama de componentes da configuração do sistema**

## 6. Conclusões

Este artigo tratou do problema de projeto de sistemas embarcados. Apresentamos os conceitos básicos da metodologia *Application-Oriented System Design* (AOSD), que foi desenvolvida para tratar desse problema, e focamos na parte final desse desenvolvimento, uma ferramenta auxiliar o projetista no processo de configuração e geração do sistema. Como estudo de caso esse processo foi utilizado para a configuração de geração uma

aplicação real. Embora seja uma aplicação simples, ela demonstra a facilidade de uso de tal ferramenta e sua potencialidade na geração de sistemas mais complexos. Técnicas de engenharia de software garantem a reusabilidade e portabilidade do sistema, enquanto técnicas de implementação, como metaprogramação estática, garantem a resolução de várias dependências em tempo de compilação com overhead mínimo (ou mesmo inexistente).

Assim, a estratégia desenvolvida é uma solução abrangente, prática e eficiente. A implementação atual ainda não representa uma solução completa para automatizar o processo de geração de sistemas embarcados, possivelmente baseados em SoCs, pois os processos automáticos ainda estão em desenvolvimento. Entretanto, métodos independentes podem ser suportados em cada etapa, sem afetar a estrutura ou a implemetação do resto da ferramenta.

Assim, os trabalhos futuros, e mesmo em andamento, incluem principalmente o desenvolvimento de métodos interativos e automáticos para a geração de várias alternativas de projeto e a avaliação das características e custos desses projetos para a escolha da alternativa mais indicada. Esses métodos correspondem à exploração do espaço de projeto e particionamento hardware/software (hw/sw), etapas do projeto integrado de hw/sw (co-design). Com isso, essa abordagem poderá não apenas gerar automaticamente sistemas que atendam os requisitos de uma aplicação, mas também outros requisitos de projeto, como custo máximo, tamanho da placa de CI, ou consumo de energia.

## References

Björklund, D. and Lilius, J. (2002). From uml behavioral descriptions to efficient synthetizable vhdl. In *IEEE NORCHIP Conference*.

Cardoso, J. and Neto, H. (1999). Macro-based hardware compilation of java bytecodes into a dynamic reconfigurable computing system. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 2 – 11, Napa Valley, CA.

D. Herrmann, J. H. and Ernst, R. (1994). An approach to the adaptation of estimated cost parameters in the cosyma system. In *International Conference on Hardware Software Codesign - Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 100 – 107.

Danillo Santos, Rafael Cancian, R. d. M. and Fröhlich, A. A. (2006). Advantages and disadvantages of application-oriented system design in embedded systems design. In *Proceedings of 4th International IEEE Conference on Industrial Informatics*, pages 904 – 909, Cingapura.

et al, A. S. (2000). Optimal hardware/software partitioning for concurrent specification using dynamic programming. In *Thirteenth International Conference on VLSI Design*, pages 110 – 113.

et al, P. A. (2003). Hardware-software partitioning in embedded system design. In *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*.

et al, P. E. (1997). System level hardware/software partitioning based on simulated annealing and tabu search. In *Design Automation for Embedded Systems*.

Fröhlich, A. A. (2001). *Application-Oriented Operating Systems*. PhD thesis, GMD - Forschungszentrum Informationstechnik, Sankt Augustin, Germany.

Fröhlich, A. A. and Schröder-Preikschat, W. (2000). Scenario adapters: Efficiently adapting components. In *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, USA.

G. Mcgregor, B. Einloth, F. V. and Stitt, G. (2005). Hardware/software partitioning of software binaries: a case study of h.264 decode. In *Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 285 – 290.

Galloway, D. (1995). The transmogrifier c hardware description language and compiler for fpgas. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 136 – 144.

Graphics, M. Catapultc. http://www.mentor.com.

Ku, D. and DeMicheli, G. (1990). Hardwarec – a language for hardware design (version 2.0). Technical report, Stanford, CA, USA.

Niemann, R. and Marwedel, P. (1996). An algorithm for hardware/software partitioning using mixed integer linear. In *Proceedings of the ED TC*.

P. Mudry, G. Z. and Tempesti, G. (2006). Hybrid genetic algorithm for constrained hardware-software partitioning. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 1 – 6.

Parnas, D. L. (1976). On the design and development of pro- gram families. In *IEEE Transactions on Software Engineering*, pages 1–9.

Polpeta, F. V. and Fröhlich, A. A. (2004). Hardware mediators: a portability artifact for component-based systems. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, volume 3207 of LNCS, Aizu,Japan.

Stitt, G. and Vahid, F. (2002). Hardware/software partitioning of software binaries. In *IEEE/ACM International Conference on Computer Aided Design*.

Tondello, G. F. and Fröhlich, A. A. (2004). Configuration management of embedded operating systems using application-oriented system design. In *Proceedings of the 5th Argentine Symposium on Computing Technology (part of the 33rd Argentine Conference on Computer Science and Operational Research)*, Córdoba, Argentine.

Tondello, G. F. and Fröhlich, A. A. (2005). On the automatic configuration of application-oriented operating systems. In *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications*, pages 120 – 123, Cairo, Egypt.

V. Srinivasan, S. R. and Vemuri, R. (1998). Hardware software partitioning with integrated hardware design space exploration. In *Proceedings of Design, Automation and Test in Europe*, pages 28 – 35.