Plataforma computacional *open-source* e de arquitetura aberta para análise de sinais biomédicos

Juliano J. Duque[1], **Luiz E. V. Silva**[1], **Luiz O. Murta Junior**[1]

[1] Grupo de Computação em Sinais e Imagens Médicas
Departamento de Física e Matemática
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto
Universidade de São Paulo

julianojd@gmail.com, luizeduardovs@gmail.com, murta@ffclrp.usp.br

Abstract. *Dynamical analysis upon biomedical signals are important representations of the physiological state in tissues, organs, or even entire human body. Therefore, much attention is devoted to the study of analysis methods that helps to extract the largest amount of relevant information from these data. This paper presents an open-source and open-architecture software platform for biomedical signal analysis, called JBioS. Implemented in Java, in addition to providing resources for data handling and pre-processing, it easily allows a rapid implementation and integration of new computational functionalities or methods through plugins, promoting validation process of new analysis methods. With these features, JBioS presents itself as a tool with potential applications in both research and clinical settings.*

Resumo. *Análises sobre a dinâmica dos sinais biomédicos são representações importantes do estado fisiológico de tecidos, orgãos, ou mesmo todo o corpo humano. Por isso, uma grande atenção é voltada para o estudo de métodos de análise que auxiliam na extração da maior quantidade de informações relevantes a partir destes dados. Este trabalho apresenta uma plataforma de software open-source e de arquitetura aberta para análise de sinais biomédicos, denominada JBioS. Implementada na linguagem Java, além de prover algumas facilidades para manipulação e pré-processamento dos dados, ela permite a implementação e integração fácil e rápida de novos métodos e funcionalidades computacionais através de plugins, promovendo o processo de validação de novos métodos de análise. Com essas características, a JBioS apresenta-se como uma ferramenta com potencial de aplicação tanto na pesquisa como no contexto clínico.*

1. Introdução

Em diversas áreas onde há presença de sistemas dinâmicos, estudos são conduzidos através do processamento e análise de séries temporais ou sinais provenientes de seus sistemas de interesse. No contexto da medicina, por exemplo, é esta análise que frequentemente permite uma melhor avaliação do quadro clínico de pacientes. Vários são os exames aplicados em pacientes que têm como resultado final um sinal biomédico, ou conjunto deles. Tais sinais são séries temporais de medidas, invasivas ou não, obtidas do organismo humano, que representam a atividade dos sistemas fisiológicos e possibilitam a identificação

de situações normais e patológicas. Com a ajuda destes resultados, os médicos especialistas podem tomar as medidas necessárias e mais adequadas, viabilizando ao paciente a oportunidade de um tratamento também mais adequado ou um diagnóstico mais preciso.

Existem inúmeros exemplos de sinais biomédicos. O eletrocardiograma (ECG), o eletroencefalograma (EEG) e o eletromiograma (EMG) são exemplos de sinais baseados em atividade elétrica de superfície corpórea. Por outro lado, a magnetocardiografia e a magnetogastrografia são sinais baseados na atividade magnética associada a órgãos específicos. Outros exemplos de sinais que podem ser coletados a partir do corpo humano são aqueles provindos da locomoção, da respiração, da pressão arterial, do piscar de olhos, entre outros.

Com o aperfeiçoamento progressivo dos computadores, ampliando cada vez mais a capacidade de processamento e armazenamento computacional, o processamento destes sinais por software vem ganhando mais força e destaque devido sua maior aplicabilidade e facilidade de implementação, comparado ao processamento por hardware.

Neste contexto, a proposta da plataforma JBioS é oferecer um ambiente de software *open-source* para a análise de dados biomédicos e novos métodos computacionais, que provê portabilidade e extensibilidade através do suporte a integração de métodos de processamento e análise de sinais por meio de *plugins*. Existem ainda outras propostas de software para processamento e análise de sinais biomédicos, porém com domínios de aplicação específicos, sendo também, em sua maioria, desenvolvidos em arquitetura e códigos fechados.

Com esta proposta espera-se colaborar, tanto no meio clínico como no de pesquisa, com este importante campo da ciência que envolve o estudo de sinais biomédicos.

2. Metodologia

2.1. Visão Geral

Como ilustrado na Figura 1, a análise de sinais pode ser resumida, dentro do contexto de um aplicativo de software, em um processo que segue sequencialmente as etapas de seleção (entrada) de sinais, possíveis pré-processamentos, processamento e análise principais, e geração e visualização de resultados para posterior avaliação.

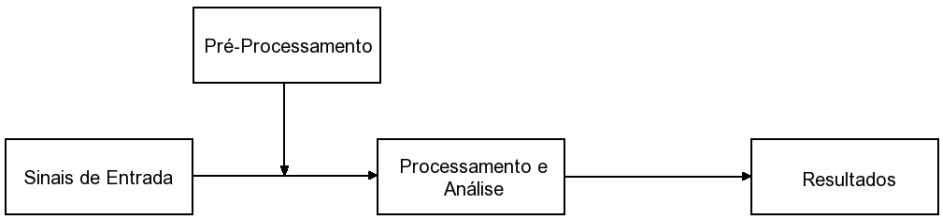

Figura 1. Esquema geral de um processo de análise de sinais, mostrando as etapas constituintes.

Baseado neste modelo, a plataforma JBioS foi desenvolvida buscando agrupar logicamente estas etapas gerais identificadas. Esta divisão lógica pode ser observada tanto na interface gráfica quanto na elaboração das classes que compõem o sistema.

A interface gráfica está definida em uma janela principal que apresenta painéis diferentes para cada uma das etapas gerais (Figura 2). Elas são representadas pelos painéis *Signals*, *Analyse* e *Results*, que são acessados por meio de abas presentes na barra superior da janela.

Quanto à estrutura, a plataforma foi construída utilizando o paradigma de orientação a objetos e as classes criadas foram dispostas em diferentes pacotes. A partir do pacote raiz foram definidos os subpacotes *analysis*, *gui*, *math* e *signal*. Em *gui* estão as classes que constroem as interfaces gráficas; em *signal*, as que representam os sinais e as formas de entrada; em *analysis*, as relacionadas com os métodos de análises, inclusive os *plugins*; e em *math*, as que realizam cálculos utilizados por métodos de análise.

Objetivando a portabilidade em diferentes sistemas operacionais, o desenvolvimento do JBioS foi feito em linguagem Java, através da versão 6 da Java Platform, Standard Edition (JavaSE 6), utilizando o ambiente de desenvolvimento integrado Netbeans IDE, disponibilizados pela Sun Microsystems. Os recursos gráficos para exibição, tanto dos sinais quanto dos resultados das análises, foram obtidos a partir da biblioteca *open-source* JFreeChart [Gilbert and Morgner 2010], que permite a criação de gráficos de boa qualidade.

2.2. Descrição das Funcionalidades

2.2.1. Sinais de Entrada

Em estudos que buscam validar métodos de análise, somente o uso de dados reais, como sinais obtidos de pacientes em procedimentos clínicos, torna o processo de validação mais difícil. Usar sinais simulados permite avaliar o desempenho dos métodos em situações nas quais se conhece as características do sinal. Assim, a plataforma fornece diferentes opções quanto ao formato dos sinais de entrada, além de prover a simulação de sinais com características bem conhecidas.

Os sinais que podem ser gerados a partir de simulações são: ruído branco (uma série de valores aleatórios, com distribuição uniforme), ruído 1/f (um fenômeno ubíquo na natureza, cujo espectro de potência é a função 1/f), mapa logístico [May 1976], mapa de Hénon [Hénon 1976] e mapa quadrático [Grebogi et al. 1983].

Além dos sinais simulados, a plataforma permite a importação de sinais através de arquivos textos, em que cada arquivo pode ter um ou mais canais dispostos em colunas e um pequeno cabeçalho padronizado contendo mínimas informações relevantes. O formato de arquivo do protocolo de comunicação SCP-ECG [CEN 2007] para sinais eletrocardiográficos também pode ser usado para fornecer sinais de entrada para o sistema. Para implementação deste módulo foi utilizado o suporte para leitura de arquivos deste formato do software PixelMed Java DICOM Toolkit, PixelMed Publishing [Clunie 2009].

O sistema também oferece uma funcionalidade que permite a leitura de sinais diretamente da base de dados do Physionet [Goldberger et al. 2000], que disponibiliza uma grande quantidade de dados de variados tipos de exames médicos. Esta implementação baseou-se nas funções oferecidas pela biblioteca WFDB Physionet, desenvolvida na linguagem C, e foi realizada através da criação de uma biblioteca nativa dinâmica para leitura destes sinais.

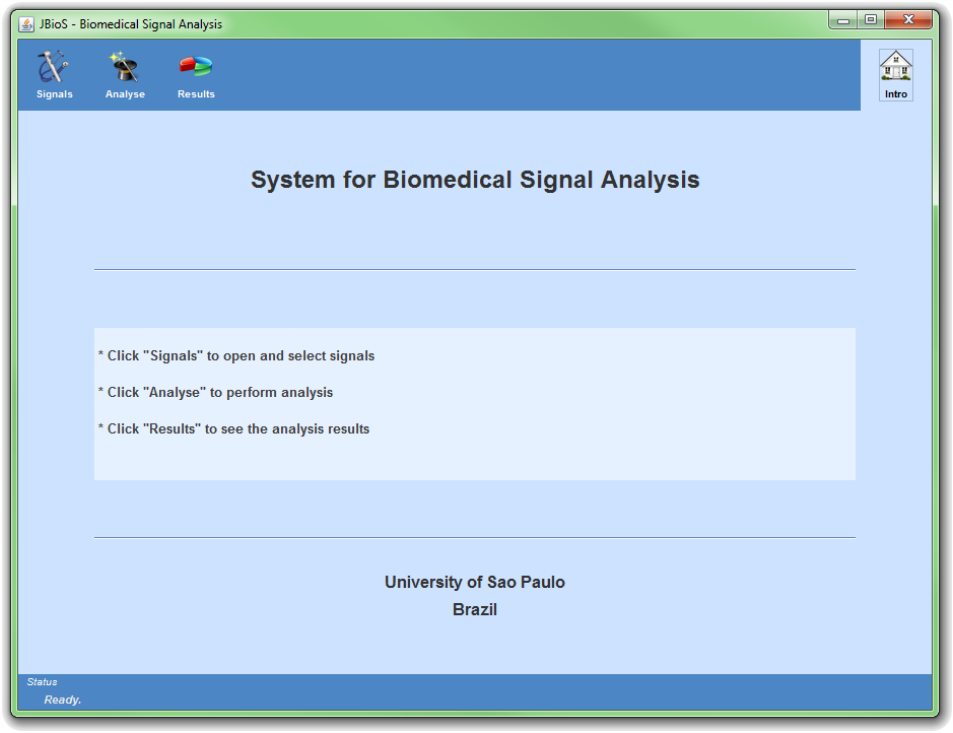

Figura 2. Painel de abertura do sistema, mostrando as opções iniciais: *Signals* para as fontes de sinais simulados, de arquivos, ou via rede; *Analyse* para iniciar uma análise; e *Resultados* provendo acesso aos resultados.

2.2.2. Ferramentas de Pré-processamento

Algumas ferramentas que podem ser usadas como operações de pré-processamento já estão implementadas, como detector de picos com possibilidade de correção manual, análise de componentes independentes (ICA), inversão de polaridade e seletor de trecho. São operações que podem ser aplicadas sobre os sinais antes que eles sejam submetidos ao processo de análise.

Para o detector de picos são propostas três implementações [So and Chan 1997]: *RPeaks*, que usa uma abordagem de *threshold* fixo; *RPoints* e *MaximumSlope*, que são abordagens de *threshold* adaptativo. A análise de componentes independentes (ICA) está presente através do algoritmo FastICA [Hyvärinen 1999] .

2.2.3. Métodos de Análise

Os métodos de análise são oferecidos sob duas perspectivas: métodos nativos da plataforma e *plugins* adicionados pelo usuário.

Dentro dos nativos estão compreendidos tanto alguns métodos de análise linear quanto não linear, sendo eles: Parâmetros no Domínio do Tempo [Malik et al. 1996], Transformada de Fourier [Haykin and Veen 1998], Auto Regressão [Shiavi 2007], Entropia Aproximada e Amostral [Pincus 1991, Richman and Moorman 2000], Entropia Multiescala [Costa et al. 2002], Transformada q-Fourier [Umarov et al. 2006], e Análise Sur-

rogate [Theiler et al. 1992], sendo que esta última utiliza a Entropia Amostral como discriminante estatístico. Todos os métodos de entropia e também a transformada q-Fourier foram implementados utilizando o paradigma não extensivo de Tsallis [Tsallis 1988], de maneira que nesses métodos surge o parâmetro q de não extensividade, e para $q = 1$ as formulações clássicas originais são recuperadas [Silva 2010].

Para a funcionalidade de *plugins*, foi projetada e desenvolvida uma *interface* que permite a interação destes com a plataforma. Os métodos nativos também foram implementados usando esta estrutura. A possibilidade de adicionar novos componentes de software à plataforma garante extensibilidade para o aprimoramento das suas funcionalidades.

2.2.4. Exibição dos Resultados

Os resultados das análises são apresentados por meio de gráficos, construídos com suporte da biblioteca JFreeChart. Junto aos gráficos também podem surgir tabelas para representar as medidas singulares, nas quais cada linha é formada por um rótulo para a medida, o seu valor e sua unidade. Em uma análise feita pela escolha de vários métodos, cada um deles terá seus resultados exibidos em diferentes abas dentro do painel de visualização.

A plataforma provê suporte para que o usuário consiga construir estas representações de exibição dos resultados obtidos pelos seus *plugins*. Há também como salvar os resultados de uma análise para consulta futura, além da possibilidade de exportar os gráficos como imagens no formato PNG.

2.3. Estrutura de Plugins

A característica de arquitetura aberta que a JBioS oferece permite que os usuários implementem seus próprios métodos de análise, de maneira que o sistema reconheça e disponibilize tais métodos como *plugins*. Para isso, foi definida uma estrutura que os usuários devem seguir para a criação dos *plugins*. Parte dela foi baseada em uma funcionalidade similar oferecida pelo software de processamento de imagens ImageJ [Rasband 2010], também de arquitetura aberta.

Basicamente, escrever um *plugin* consiste em criar uma classe Java que implemente a interface *AnalysisMethod*, definida no pacote *analysis*, utilizando as classes auxiliares *Signal*, *InputParametersDialog* e *AnalysisResult*. A primeira contém a representação de um sinal no sistema, que será usada para obter as amostras da série temporal e outras informações que possam ser relevantes na análise; a segunda permite obter alguns parâmetros de entrada antes da execução do método de análise; e a terceira é necessária para agrupar e retornar os resultados finais de uma análise, seja como valores singulares ou séries de valores para a formação de gráficos. A Figura 3 ilustra o diagrama de classes envolvidas no processo de criação de *plugins*.

A interface *AnalysisMethod* possui três métodos que deverão ser implementados em um *plugin*. No método ***doAnalysis*** o usuário deve implementar toda a lógica do processamento dos sinais recebidos como parâmetro, retornando um objeto *AnalysisResult* contendo todos os resultados do processamento. O método ***inputParameters*** é chamado pelo sistema ao selecionar o *plugin* na sua interface gráfica, permitindo que o usuário

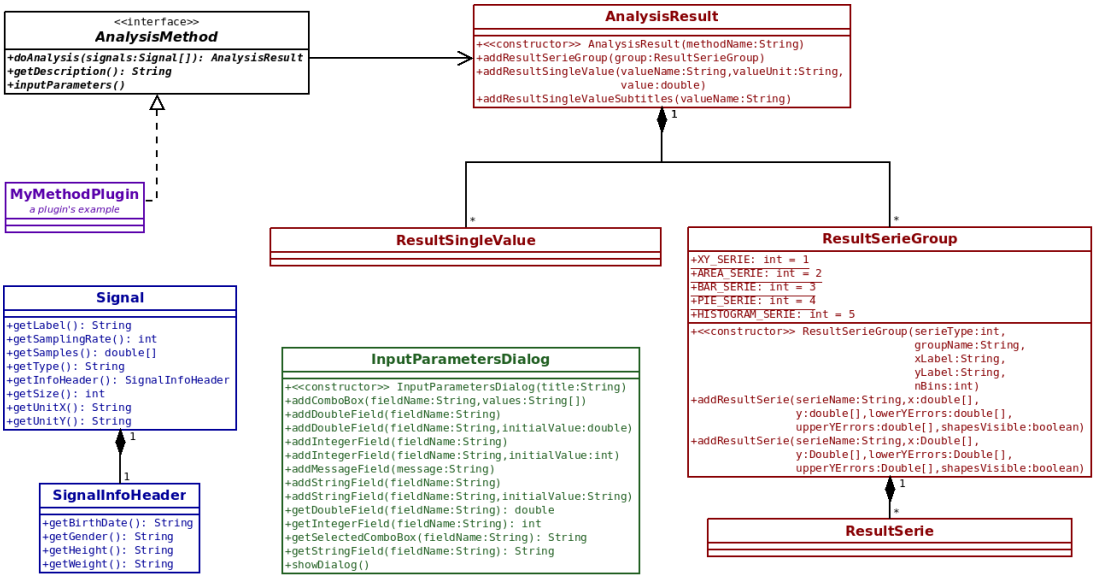

Figura 3. Diagrama de classes ilustrando as classes e métodos que são utilizados na estrutura de criação dos *plugins*. Esta estrutura tem *AnalysisMethod* como principal interface, com alguns métodos que obrigatoriamente compõem um novo *plugin*.

defina os parâmetros de entrada que serão importantes para a análise. A implementação desta parte pode ser feita utilizando a classe *InputParametersDialog*, que oferece suporte para fácil obtenção de parâmetros. O método ***getDescription*** deve retornar um objeto *String* contendo uma breve descrição do *plugin*, que será exibida na interface gráfica.

A classe *AnalysisResult* agrupa todos os resultados obtidos em dois grupos principais: resultados singulares e grupo de séries. O primeiro grupo de resultados é mostrado em uma tabela, e o segundo em gráficos. Esses grupos de séries podem ser de cinco tipos diferentes: séries XY, séries com área preenchida, gráfico de barras, gráfico de pizza ou histograma. Esta distinção permite ao sistema identificar os tipos de resultado e exibí-los de maneira apropriada.

O nome definido para um *plugin* deve terminar com um caracter *underscore* (“_”), de modo que o sistema possa diferenciar as classes principais de classes auxiliares. Se um *plugin* for composto de mais de uma classe é recomendado agrupar todas as classes em um diretório e colocá-lo na pasta “plugins”. O sistema reconhece os *plugins* localizados nesta pasta e nos diretórios de primeiro nível.

Todos os métodos de análise do sistema (nativos e *plugins*) utilizam a mesma estrutura de implementação descrita acima, e sua execução é disparada através de *threads*, o que permite a execução paralela de vários métodos. Isto impede que a interface gráfica congele e permite que operações como pré-processamento, visualização de outros resultados e entrada de novos sinais possam ser feitas enquanto a análise está em execução. Para isso, o sistema é dotado de um gerenciador que identifica os métodos selecionados, dispara sua execução e aguarda pelo término de todos, para somente então retornar os resultados e exibí-los.

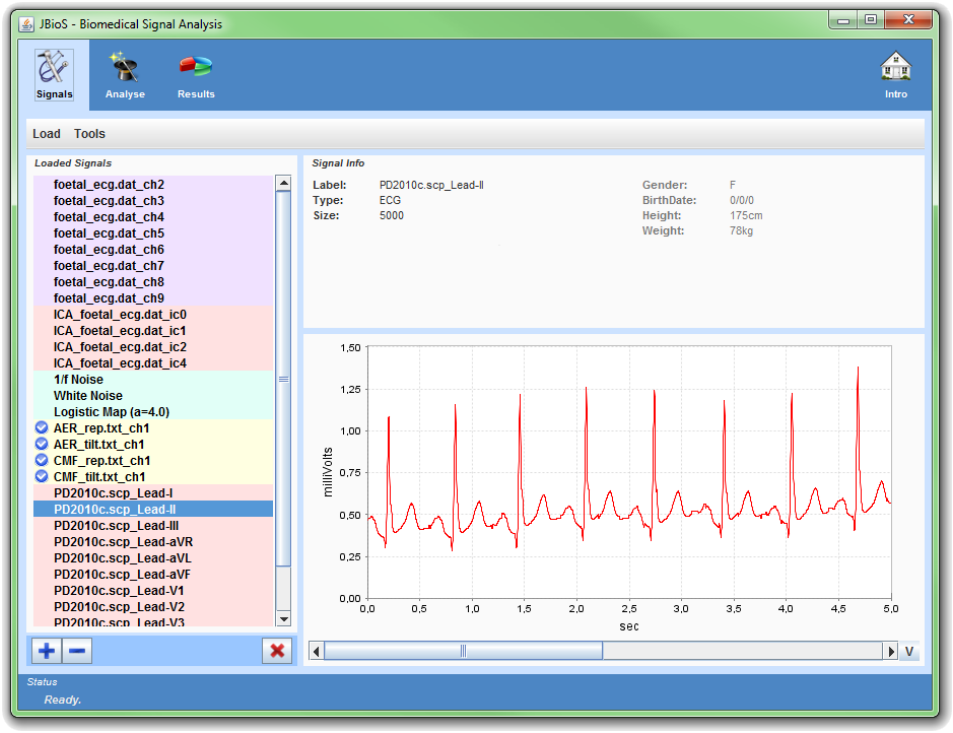

Figura 4. Painel de manipulação dos sinais, ilustrando as possibilidades de fontes de sinais implementadas no sistema. Estas possibilidades englobam sinais de arquivos, incluindo o padrão SCP-ECG; sinais simulados; e sinais de banco remoto, via rede.

3. Resultados

A Figura 2 mostra uma imagem da tela de abertura do sistema. O painel superior contém botões que simulam um componente gráfico de abas. Além da aba *Intro* estão presentes as abas *Signals*, *Analyse* e *Results*, sendo que cada uma representa uma das etapas gerais dentro do processo de análise de um sinal. O painel intermediário exibe o conteúdo da aba selecionada, e o painel inferior é a barra de status que indica as ações sendo executadas, informando ao usuário o estado em que se encontra o sistema.

A Figura 4 mostra o painel *Signals*. Este painel é dedicado ao carregamento, visualização e seleção de sinais, além da execução de pré-processamentos que visam adequar um sinal para a fase de análise. O painel apresenta uma barra de menus, onde se encontram as opções de entrada de sinais e pré-processamento. À esquerda há uma lista, onde são exibidos os sinais carregados, com botões na parte inferior. Estes botões controlam a seleção de sinais que serão submetidos para etapa de análise, bem como a exclusão deles da lista de sinais carregados. À direita está o painel de informações contendo, entre outros, o rótulo, o tipo, e o tamanho do sinal selecionado; abaixo dele está o painel que exibe a representação gráfica do sinal. Selecionar um dos sinais da lista exibe suas informações e sua representação gráfica nestes painéis da direita.

A Figura 5 mostra o painel *Analyse*. A funcionalidade deste painel é permitir ao usuário escolher, configurar e executar os métodos de análise sobre os sinais selecionados para este fim. Ele é formado por uma tabela, que mostra informações dos sinais selecionados para análise; dois conjuntos de caixas de seleção (um de métodos nativos e outro

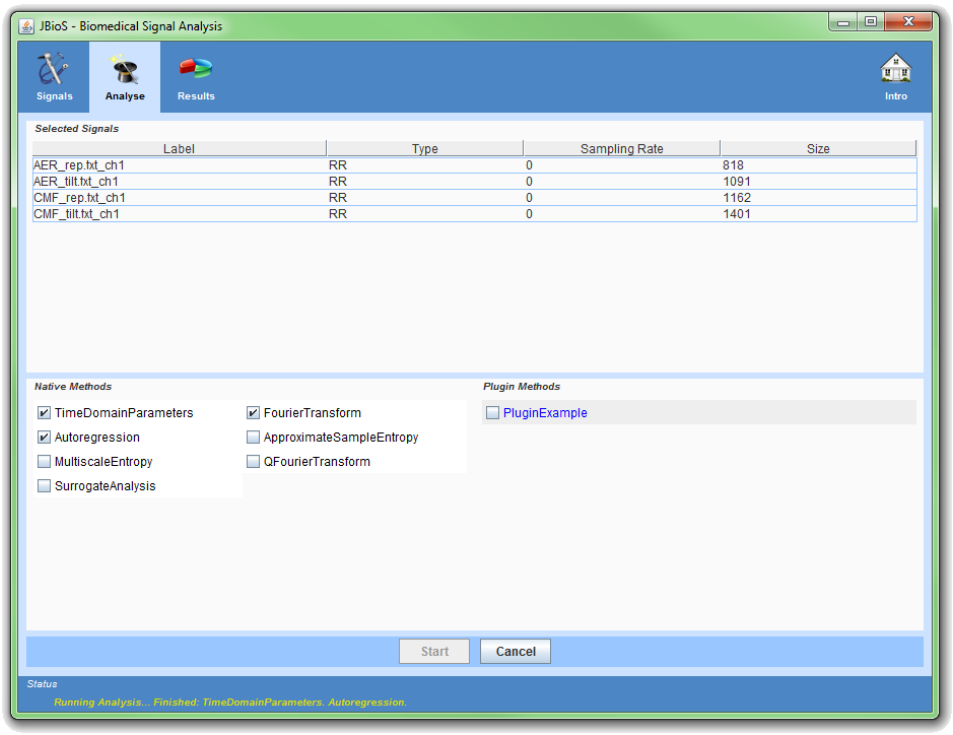

Figura 5. Painel de seleção dos métodos de análise, onde estão disponíveis os *plugins* nativos do sistema (em preto) e os adicionados pelo usuário (em azul).

de *plugins*), que ao serem marcadas exibem caixas de diálogo para configurar possíveis parâmetros usados pelo método; e botões para iniciar e cancelar o processo de análise. Ao iniciar o processo de análise todos os métodos entram em execução. O cancelamento termina a execução de todos eles e nenhum resultado é gerado.

A Figura 6 mostra o painel *Results*. Este painel apresenta os resultados e é composto por uma lista a esquerda, que inclui as análises realizadas; uma barra com botões para salvar e apagar resultados de análises recém obtidas e para abrir outras salvas anteriormente; e o painel para exibição dos gráficos e demais informações dos resultados.

Os resultados obtidos para cada análise já realizada são agrupados e referenciados por um item na lista. Os gráficos e demais informações de cada análise são separados em abas, cada uma contendo os resultados de um método, e visualizados no painel à direita.

4. Conclusões e Perspectivas

A plataforma apresentada neste trabalho é uma proposta de ferramenta extensível e *open-source* com domínio de aplicação no campo da pesquisa e também da prática clínica. Sendo *open-source*, as funcionalidades básicas, como por exemplo as ferramentas de entrada de sinais e de pré-processamento, podem ser progressivamente melhoradas pela comunidade; e sendo uma plataforma com arquitetura aberta, novos métodos de análise de sinais podem ser facilmente incorporados como *plugins*, exigindo o mínimo de desenvolvimento por parte dos usuários, e consequentemente promovendo o aumento do número de métodos disponíveis para a plataforma. Assim, a plataforma JBioS apresenta um grande potencial de aplicação, podendo contribuir significativamente nas áreas que abrangem a análise de sinais biomédicos.

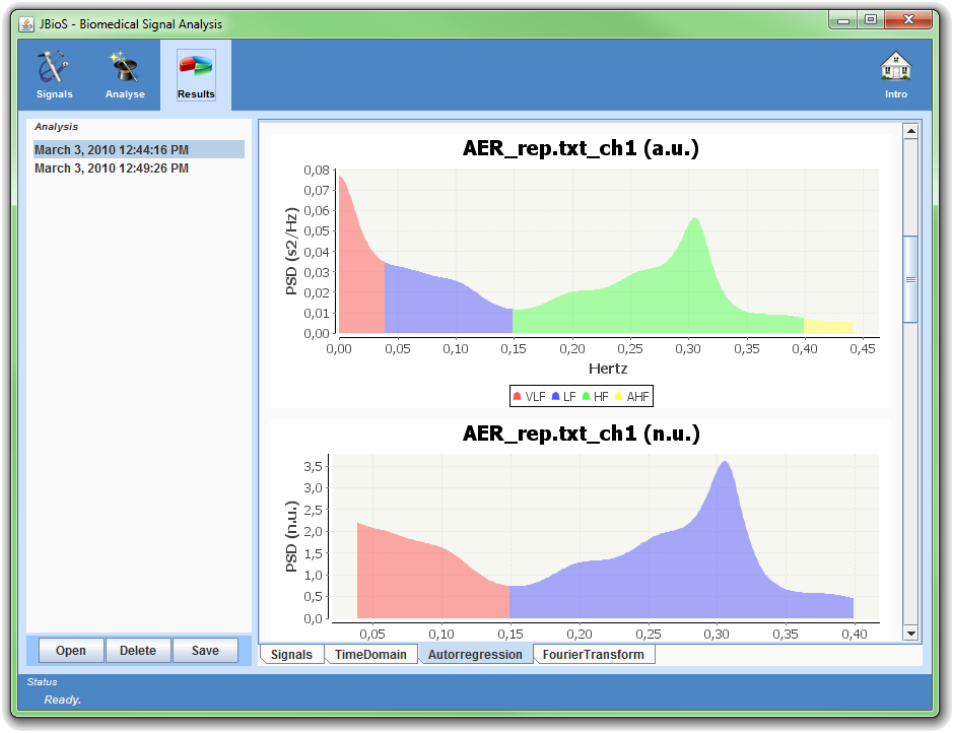

Figura 6. Painel de exibição dos resultados, mostrando abas que possibilitam a visualização de diferentes resultados.

Essa ferramenta foi idealizada e desenvolvida no Laboratório de Computação em Sinais e Imagens Médicas (CSIM) da USP, onde já vem sendo utilizada em alguns trabalhos de pesquisa, o que contribuiu para a identificação de problemas, correção e ampliação das suas funcionalidades. Outras melhorias que estão sendo planejadas incluem o desenvolvimento de outros tipos de sinais simulados e ferramentas de pré-processamento.

Espera-se em breve disponibilizar o acesso a esta plataforma para a comunidade em algum repositório público de amplo acesso.

5. Agradecimentos

Os autores agradecem a FAPESP pelo apoio financeiro durante o desenvolvimento deste trabalho.

Referências

CEN (2007). Standard Communications Protocol for Computer-Assisted Electrocardiography. European Pre-Standard ENV 1064.

Clunie, D. A. (2009). PixelMed Java DICOM Toolkit. http://www.pixelmed.com.

Costa, M., Goldberger, A. L., and Peng, C.-K. (2002). Multiscale entropy analysis of complex physiologic time series. *Phys. Rev. Lett.*, 89(6):068102–.

Gilbert, D. and Morgner, T. (2010). Jfreechart project. http://www.jfree.org/jfreechart.

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000). Physiobank,

physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220.

Grebogi, C., Ott, E., and Yorke, J. A. (1983). Crises, sudden changes in chaotic attractors, and transient chaos. *Physica D Nonlinear Phenomena*, 7:181–200.

Haykin, S. S. and Veen, B. V. (1998). *Signals and Systems*. John Wiley & Sons, Inc., New York, NY, USA.

Hyvärinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634.

Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics*, 50:69–77.

Malik, M., Bigger, J. T., Camm, A. J., Kleiger, R. E., Malliani, A., Moss, A. J., and Schwartz, P. J. (1996). Heart rate variability: Standards of measurement, physiological interpretation, and clinical use. *Circulation*, 93(5):1043–1065.

May, R. M. (1976). Simple mathematical models with very complicated dynamics. *Nature*, 261:459–467.

Pincus, S. M. (1991). Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences of the United States of America*, 88(6):2297–2301.

Rasband, W. S. (1997-2010). ImageJ, U. S. National Institutes of Health, Bethesda, Maryland, USA. http://rsb.info.nih.gov/ij.

Richman, J. S. and Moorman, J. R. (2000). Physiological time-series analysis using approximate entropy and sample entropy. *Am J Physiol Heart Circ Physiol*, 278(6):H2039–2049.

Shiavi, R. (2007). Random signal modeling and parametric spectral estimation. In *Introduction to Applied Statistical Signal Analysis*, pages 287 – 330. Academic Press, Burlington, third edition edition.

Silva, L. E. V. (2010). Ferramentas computacionais na análise da variabilidade da frequência cardíaca através do paradigma não extensivo no estudo de cardiopatias. Master's thesis, Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Universidade de São Paulo.

So, H. and Chan, K. (1997). Development of qrs detection method for real-time ambulatory cardiac monitor. In *Engineering in Medicine and Biology Society, 1997. Proceedings of the 19th Annual International Conference of the IEEE*, volume 1, pages 289–292.

Theiler, J., Eubank, S., Longtin, A., Galdrikian, B., and Doyne Farmer, J. (1992). Testing for nonlinearity in time series: the method of surrogate data. *Physica D Nonlinear Phenomena*, 58:77–94.

Tsallis, C. (1988). Possible generalization of boltzmann-gibbs statistics. *Journal of Statistical Physics*, 52:479–487.

Umarov, S., Tsallis, C., and Steinberg, S. (2006). A generalization of the central limit theorem consistent with nonextensive statistical mechanics.