Título em Português:	DESENVOLVIMENTO E IMPLEMENTAÇÃO DE INTERFACE DE VALIDAÇÃO PARA MÉTODOS DE RESSONÂNCIA MAGNÉTICA
Título em Inglês:	DEVELOPMENT AND IMPLEMENTATION OF A VALIDATION INTERFACE FOR MAGNETIC RESONANCE METHODS
Autor:	Andre Rodrigues de Freitas Batista
Instituição:	Universidade Virtual do Estado de São Paulo
Unidade:	Instituto de Física de São Carlos
Orientador:	Alberto Tannus
Área de Pesquisa / SubÁrea:	Tecnologia e Inovação
Agência Financiadora:	Outros

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE INTERFACE DE VALIDAÇÃO PARA MÉTODOS DE RESSONÂNCIA MAGNÉTICA

Andre Rodrigues de Freitas Batista[1,2]

Daniel Cosmo Pizetta[2]

Alberto Tannús[2]

[1]UNIVESP, [2]CIERMag - IFSC - USP

andrerfbatista@gmail.com

Objetivos

O objetivo deste trabalho é introduzir a implementação de testes e interações por meio de uma interface em linha de comando (CLI) para o *framework* genérico para programação de Ressonância Magnética PyMR [1]. Adicionalmente, busca-se implementar também a interface gráfica após a validação de métodos/projetos e seus respectivos protocolos. A validação de projetos/métodos é uma forma de garantir a qualidade no código final e permitir, de forma automatizada, a execução de testes de software.

Métodos e Procedimentos

A implementação utilizou Python 3.11 como linguagem de programação em conjunto com bibliotecas como Logging, Argparse, Subprocess, JSON e PyQt5/Qt5 para interface gráfica. Foi implementada a interação por meio de CLI com *parser* para validar os projetos/métodos de maneira individual ou coletiva, nas áreas do sistema ou usuário, executando todas as validações necessárias. Os resultados das validações são apresentados no terminal e juntamente com as estatísticas da execução. Conforme ilustrado na figura 1, devem ser passados os argumentos para que sejam executados no terminal, sendo obrigatório o nome do projeto. Se houver erros na digitação dos argumentos, uma mensagem informará qual problema ocorreu na execução do parser. Para verificar todos os métodos, deve-se adicionar *all_projects* com as opções desejadas.

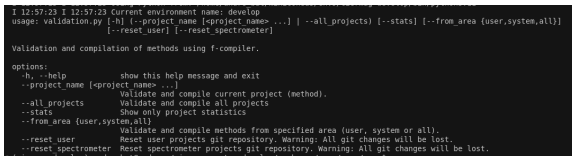

Figura 1: Interface por linha de comando da validação mostrando a documentação de ajuda.

Como forma de integrar com a interface gráfica do framework PyMR, foi adicionada uma janela chamada *Problems* implementada com Qt5/PyQt5. Após o carregamento do projeto, uma validação é executada simultaneamente e, se houver falhas, um ícone alerta o usuário em qual protocolo ocorreu a falha e quantidade, também informando sobre os problemas na validação.

Resultados

As implementações descritas permitem a validação de projetos e métodos através de uma CLI, facilitando a interação do usuário e a obtenção de resultados de validação diretamente no terminal. As informações recebidas incluem validação de protocolos, taxa de sucesso, método testado, a quantidade total de protocolos e quantos foram validados, conforme a figura 3 apresenta. Além disso, a adição de um retorno gráfico com ícones facilita

a identificação de erros, tornando o processo de validação eficiente e visualmente mais acessível. A figura 4 e 5 demonstra o PyMR com a janela *Problems* com aviso de erros no projeto.

a)

```
I 09:33:36 I 09:33:36   Statistics (Project name: CPMG, area:system):
I 09:33:36 I 09:33:36   Total methods: 1
I 09:33:36 I 09:33:36   Validated methods: 1
I 09:33:36 I 09:33:36   Failed methods: 0
I 09:33:36 I 09:33:36   Success rate: 100.00%
I 09:33:36 I 09:33:36   Failed methods: []

I 09:33:36 I 09:33:36   Methods errors: {}
```

b)

```
Progress: |                                                  | 100% completed

I 09:49:48 I 09:49:48   Statistics for area user):
I 09:49:48 I 09:49:48   Total methods: 30
I 09:49:48 I 09:49:48   Validated methods: 19
I 09:49:48 I 09:49:48   Failed methods: 11
I 09:49:48 I 09:49:48   Success rate: 63.33%
I 09:49:48 I 09:49:48   Methods errors: {
    "CPMG_E": {
        "default_mn_1p5.json": {
            "compile": "ERROR: Non-existent variable in the symbol table: C_Synth
```

Figura 3: Resultado da interface por linha de comando. a) Projeto testado sem nenhuma falha. b) Todos os projetos da área 'user' testados com relatório em JSON.

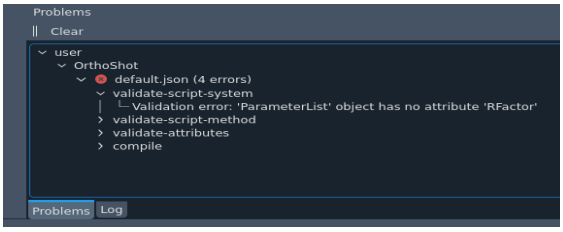

Figura 4: Implementação da interface de validação na janela *Problems*. A janela lista os projetos do usuário, neste caso *OrthoShot*, onde foi validado o protocolo *default.json*. Os itens de validação falhos são mostrados com suas respectivas mensagens de erros.

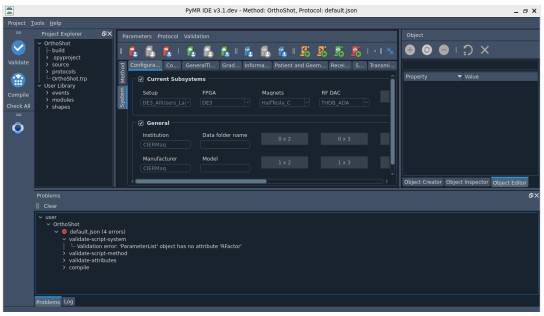

Figura 5: Visão geral do IDE [2] do PyMR e a integração com a janela *Problems* mostrando a validação do projeto/método corrente.

Conclusões

A integração de uma interface CLI para o software PyMR, juntamente com a visualização gráfica de erros, melhora significativamente a experiência do usuário e a eficácia do processo de validação de métodos e projetos. As Implementações descritas proporcionam uma maneira intuitiva e eficiente de realizar testes e obter *feedback* visual imediato, o que é essencial para a confiabilidade e usabilidade do software de Ressonância Magnética. Além disso, com a interface CLI, é possível realizar as validações automáticas/programáticas sem a necessidade de interação do usuário

Agradecimentos

Gostaria de agradecer ao meu grande amigo Daniel C. Pizetta por todos os ensinamentos, paciência, dedicação e incentivo, não só nesse trabalho, mas em várias etapas da minha vida. Agradeço ao Professor Alberto Tannús por me aceitar no projeto, mesmo eu tendo muito pouco conhecimento sobre Ressonância Magnética. Agradeço também aos amigos que fiz no grupo CIERMag.

Referências

[1] Pizetta DC. PyMR: A Framework for Programming Magnetic Resonance Systems. PhD thesis. Instituto de Física de São Carlos; 2018. DOI: 10.11606/T.76.2019.tde-06052019-103714

[2] Pizetta DC. Biblioteca, API e IDE para o desenvolvimento de projetos de metodologias de Ressonância Magnética. Master's thesis. São Carlos: Universidade de São Paulo, Instituto de Física de São Carlos; 2014. doi:10.11606/D.76.2014.tde-28042014-160738.

DEVELOPMENT AND IMPLEMENTATION OF A VALIDATION INTERFACE FOR MAGNETIC RESONANCE METHODS

Andre Rodrigues de Freitas Batista[1,2]

Daniel Cosmo Pizetta[2]

Alberto Tannús[2]

[1]UNIVESP, [2]CIERMag - IFSC - USP

andrerfbatista@gmail.com

Objectives

The objective of this work is to introduce the implementation of tests and interactions through a command-line interface (CLI) for the generic framework for Magnetic Resonance Imaging programming - PyMR [1]. Additionally, it aims to implement a graphical interface for the validation of methods/projects and their respective protocols. The validation of projects/methods is a way to ensure the quality of the final code and to allow the automated execution of software tests.

Methods and Procedures

The implementation used Python 3.11 as the programming language in conjunction with libraries such as Logging, Argparse, Subprocess, JSON, and PyQt5/Qt5 for the graphical interface. Interaction through the CLI with a parser was implemented to validate projects/methods individually or collectively, in system or user areas, executing all necessary validations. The validation results are presented in the terminal along with execution statistics. As shown in Figure 1, arguments must be passed to be executed in the terminal, with the project name being mandatory. If there are errors in typing the arguments, a message will inform what problem occurred during parser execution. To verify all methods, add all_projects with the desired options.

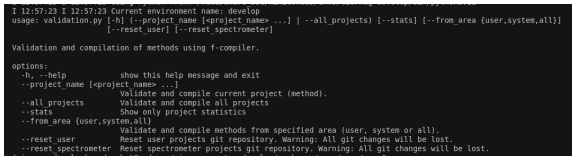

Figure 1: Command-line interface for validation showing the help documentation.

To integrate with the graphical interface of the PyMR framework, a window called Problems was added, implemented with Qt5/PyQt5. After the project is loaded, a validation is performed simultaneously, and if there are failures, an icon alerts the user to which protocol encountered the failure and the quantity, also providing information about the issues in the validation.

Results

The described implementations allow the validation of projects and methods through a CLI, facilitating user interaction and obtaining validation results directly in the terminal. The received information includes protocol validation, success rate, tested methods, total number of protocols, and how many were validated, as shown in Figure 3. Additionally, a graphical interface with icons facilitates error identification, making the validation process efficient and visually more accessible, as can be seen in figures 4 and 5 demonstrate PyMR

with the Problems window showing project error warnings.

a)

```
I 09:33:36 I 09:33:36   Statistics (Project name: CPMG, area:system):
I 09:33:36 I 09:33:36   Total methods: 1
I 09:33:36 I 09:33:36   Validated methods: 1
I 09:33:36 I 09:33:36   Failed methods: 0
I 09:33:36 I 09:33:36   Success rate: 100.00%
I 09:33:36 I 09:33:36   Failed methods: []

I 09:33:36 I 09:33:36   Methods errors: {}
```

b)

```
Progress: |                                                  | 100% completed

I 09:49:48 I 09:49:48   Statistics for area user):
I 09:49:48 I 09:49:48   Total methods: 30
I 09:49:48 I 09:49:48   Validated methods: 19
I 09:49:48 I 09:49:48   Failed methods: 11
I 09:49:48 I 09:49:48   Success rate: 63.33%
I 09:49:48 I 09:49:48   Methods errors: {
    "CPMG_E": {
        "default_mn_1p5.json": {
            "compile": "ERROR: Non-existent variable in the symbol table: C_Synth
```

Figure 3: Command-line interface results. a) Project tested without any failures. b) All projects in the user area tested with a JSON report..

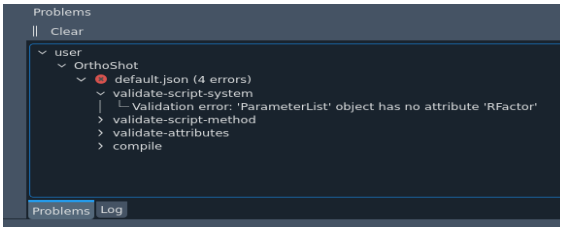

Figure 4: Implementation of the validation interface in the Problems window. The window lists the user's projects, in this case, OrthoShot, where the default.json protocol was validated. The failed validation items are shown with their respective error messages.

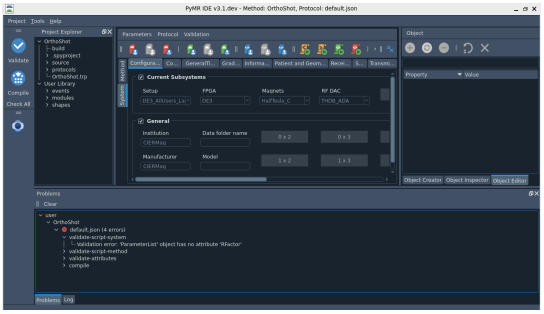

Figure 5: Overview of the PyMR IDE [2] and its integration with the Problems window showing the validation of the current project/method.

Conclusions

The integration of a CLI interface for the PyMR software, along with the graphical visualization of errors, significantly improves the user experience and the effectiveness of the method and project validation process. The described implementations provide an intuitive and efficient way to perform tests and obtain immediate visual feedback, which is essential for the reliability and usability of Magnetic Resonance Imaging software. Additionally, with the CLI interface, it is possible to perform automatic/programmatic validations without the need for user interaction

Acknowledgments

I would like to thank my great friend Daniel C. Pizetta for all the teachings, patience, dedication, and encouragement, not only in this work but throughout various stages of my life. I also thank Professor Alberto Tannús for accepting me into the project, despite my having very little knowledge about Magnetic Resonance. I also appreciate the friends I made in the CIERMag group.

References

[1] Pizetta DC. PyMR: A Framework for Programming Magnetic Resonance Systems. PhD thesis. Instituto de Física de São Carlos; 2018. DOI: 10.11606/T.76.2019.tde-06052019-103714

[2] Pizetta DC. Biblioteca, API e IDE para o desenvolvimento de projetos de metodologias de Ressonância Magnética. Master's thesis. São Carlos: Universidade de São Paulo, Instituto de Física de São Carlos; 2014. doi:10.11606/D.76.2014.tde-28042014-160738.