

Árvores de Regressão: Processo de Treinamento

Introdução ao Aprendizado de Máquina

Jean Marcelo Mira Junior
Ramiro Luiz Nunes

14 de maio de 2024

Sumário

Diferença entre Árvores de Decisão e Regressão

Para Kleppe et al. (2005), durante o desenvolvimento de software há:

- Levantamento de requisitos;
- Análise e descrição funcional;
- Elaboração do projeto;
- Codificação;
- Teste do código desenvolvido.

AID e CHART

Para Kleppe et al. (2005), durante o desenvolvimento de software há:

- Levantamento de requisitos;
- Análise e descrição funcional;
- Elaboração do projeto;
- Codificação;
- Teste do código desenvolvido.

Objetivos

- Objetivo Geral: Elaborar um procedimento para geração de código utilizando modelos UML voltado para diagramas comportamentais, especificamente o diagrama de atividades.
- Objetivos Específicos:
 - Estudar a literatura sobre transformação de modelo para texto;
 - Selecionar uma abordagem para geração de código através de diagramas comportamentais UML;
 - Projetar e implementar a geração de código;
 - Testar e analisar os resultados da implementação.

Fundamentação teórica

- Sistemas Embarcados;
- Desenvolvimento dirigido por modelos;
- Linguagem de modelagem unificada (UML);
- Geração de código.

UML e o diagrama comportamental de atividade - Nós

Figs/nosdecontrole.png

UML e o diagrama comportamental de atividade - Ações

Figs/atividadenos(2).png

Conceito de trabalho

Figs/metodo.png

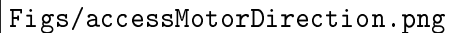
Exemplo diagrama de classe

Figs/ProxyExample.png

Modelagem no eclipse papyrus modeling

Figs/classDiagramProxy.png

Modelagem do método accessMotorDirection - Parte I lógica



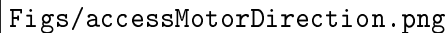
Figs/accessMotorDirection.png

Figura 6: Diagrama de atividade accessMotorDirection (Autor, 2022).

Lógica para geração de código

Figs/alg.png

Modelagem do método accessMotorDirection - Parte II lógica



Figs/accessMotorDirection.png

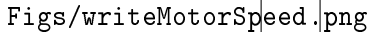
Figura 8: Diagrama de atividade accessMotorDirection (Autor, 2022).

DecisionNode e a relação com OpaqueBehavior

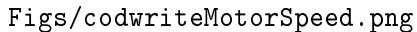
Figs/accessMotorDirection.png

Figs/codCodaccessMotorDirection.png

CreateObjectAction e CallOperationAction




(a) Diagrama de atividade



(b) Código gerado


Figura 10: Exemplo 2 de código gerado writeMotorSpeed (Autor, 2022).

ReadStructuralFeatureAction e AddStructuralFeatureValueAction



Figs/clearErrorStatus.png

(a) Diagrama de atividade



Figs/codclearErrorStatus.png

(b) Código gerado

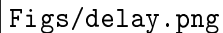
Exemplo diagrama de classe - Limitação

Figs/buttonClass.png

Modelagem no eclipse papyrus modeling - Limitação

Figs/classDiagramButton.png


Modelagem diagrama de atividade delay - Limitação



Figs/delay.png


Figura 14: Diagrama de atividade delay da classe Timer (Autor, 2022).

Comparação do código da literatura com código gerado



Figs/timingCode.png

(a) Código da literatura (Samak, 2008)



Figs/meuTimer.png

(b) Código gerado (Autor, 2022)

Conclusões

- Após estudar a literatura de transformação de modelos para texto;
- Selecionar a abordagem para geração de código embarcado através de diagrama comportamental de atividade UML;
- Realizar a implementação do gerador e obter fragmentos de código.

Conclusões

- A ferramenta consegue identificar e realizar a geração de código para os nós:
 - InitialNode;
 - DecisionNode;
 - MergeNode;
 - ActivityFinalNode.
- Além das ações:
 - ReadStructuralFeatureAction;
 - AddStructuralFeatureValueAction;
 - ActivityParameterNode;
 - ClearStructuralFeatureAction;
 - CreateObjectAction;
 - DestroyObjectAction;
 - CallOperationAction.

Conclusões

- Consequindo gerar fragmentos de código em C++ para:
 - Criar objetos;
 - Destruir objetos;
 - Atribuir valor aos objetos;
 - Retornar o valor dos objetos;
 - Estrutura If/Else;
 - Realizar a chamada de funções para os objetos.

Conclusões

- Tendo como limitações:
 - Tarefas paralelas (ForkNode e JoinNode);
 - Laço de repetição While e For (LoopNode ou DecisionNode);
 - OpaqueBehavior na condição do If;
 - Indentação.

Conclusões

- Trabalhos futuros:
 - Aumentar a quantidade de nós e ações comportamentais suportados (laços de repetição e tarefas em paralelo);
 - Controle do objeto enviado no fluxo de objetos;
 - Melhorar indentação.

Referências

- SAMEK, M. Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems. CRC Press, 2008. ISBN 9781482249262.
- KLEPPE, A.; WARMER, J.; BAST, W. MDA explained, the model driven architecture: practice and promise. 5. ed. Massachusetts: Pearson, 2005.

Fechamento

Geração de Código Usando Diagramas de Atividade para Sistemas Embarcados

Jean Marcelo Mira Junior

Orientador: Prof. Dr. Gian Ricardo Berkenbrock