Desenvolvimento Baseado em Modelos e Teste de Software

Paulo Gabriel Gadelha Queiroz

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

Sumário

Motivação

- O software deve ser não apenas confiável:
  - operar em ambientes computacionais embarcados e distribuídos;
  - comunicar-se utilizando uma grande variedade de paradigmas de comunicação;
  - se adaptar dinamicamente a mudanças nos ambientes operacionais.

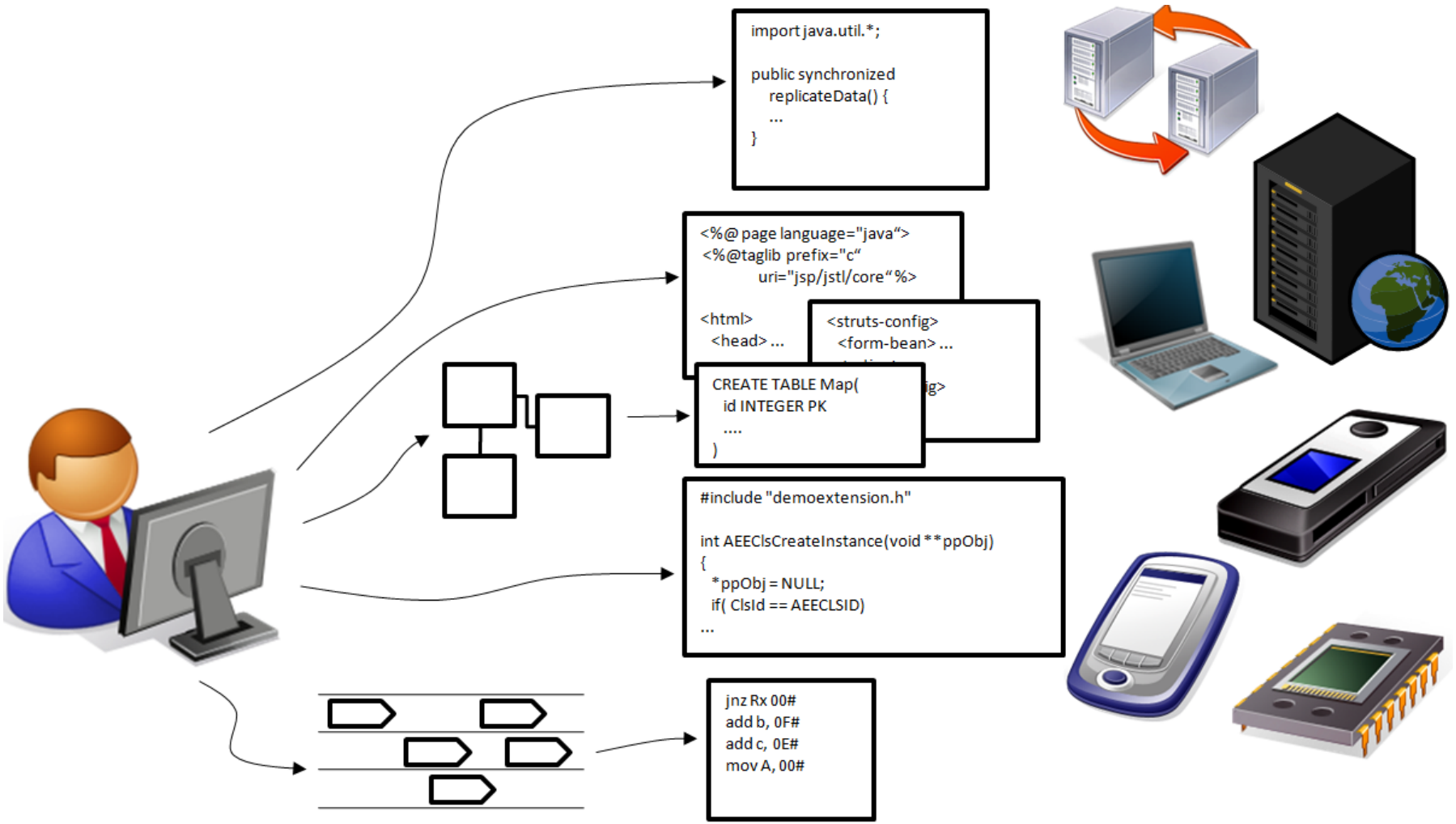
Motivação

- Software se tornou um artefato muito complexo.
- Ciclo de desenvolvimento reduzido (Time-to-Market).
- "Construir sistemas de software complexos de forma manual pode ser equiparado a construir pirâmides no antigo Egito." (France e Rumpe 2007)
- Como é desenvolvido o software tradicionalmente?
- Quais os problemas desse tipo de desenvolvimento?
- Necessidade de uma metodologia de engenharia mais disciplinada.

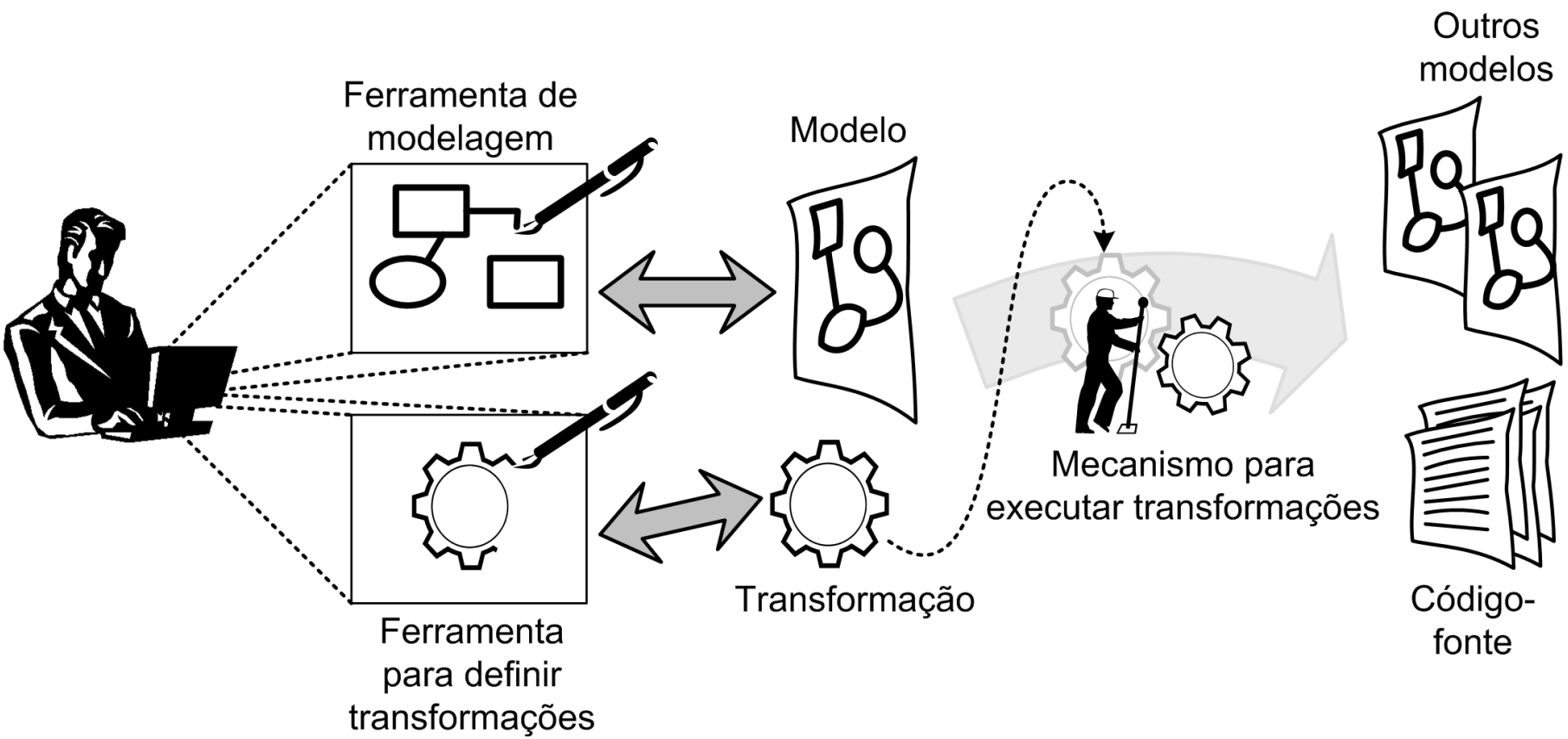
Desenvolvimento Dirigido a Modelos (MDD)

- Resolver esses problemas, enfatizando a importância de modelos no processo de desenvolvimento de software.
- No MDD, os modelos não são "apenas papel" que auxiliam nas tarefas de desenvolvimento. Eles são parte constituinte do software, assim como o código.
- Domínio do problema X Domínio da implementação/solução.
- Realização de transformações que geram artefatos de implementação automaticamente.

Desenvolvimento Tradicional



Desenvolvimento Dirigido a Modelos (MDD)



Nomenclatura

- O MDD é também conhecido como:
  - MDE (Model-Driven Engineering)
  - MDSD (Model-Driven Software Development)
- A OMG (Object Management Group) definiu uma realização particular do MDD e usou o termo Model Driven Architecture (MDA).

Engenharia Dirigida a Modelos (MDA)

- O MDA Introduz os conceitos de:
  - CIM (Computation Independet Model)-Visão do sistema de um ponto de vista não computacional.
  - PIM (Plataform Independet Model)-Visão do sistema de forma independente da plataforma de implementação.
  - PSM (Platform-Specific Model)-Visão do sistema do ponto de vista de uma plataforma específica.

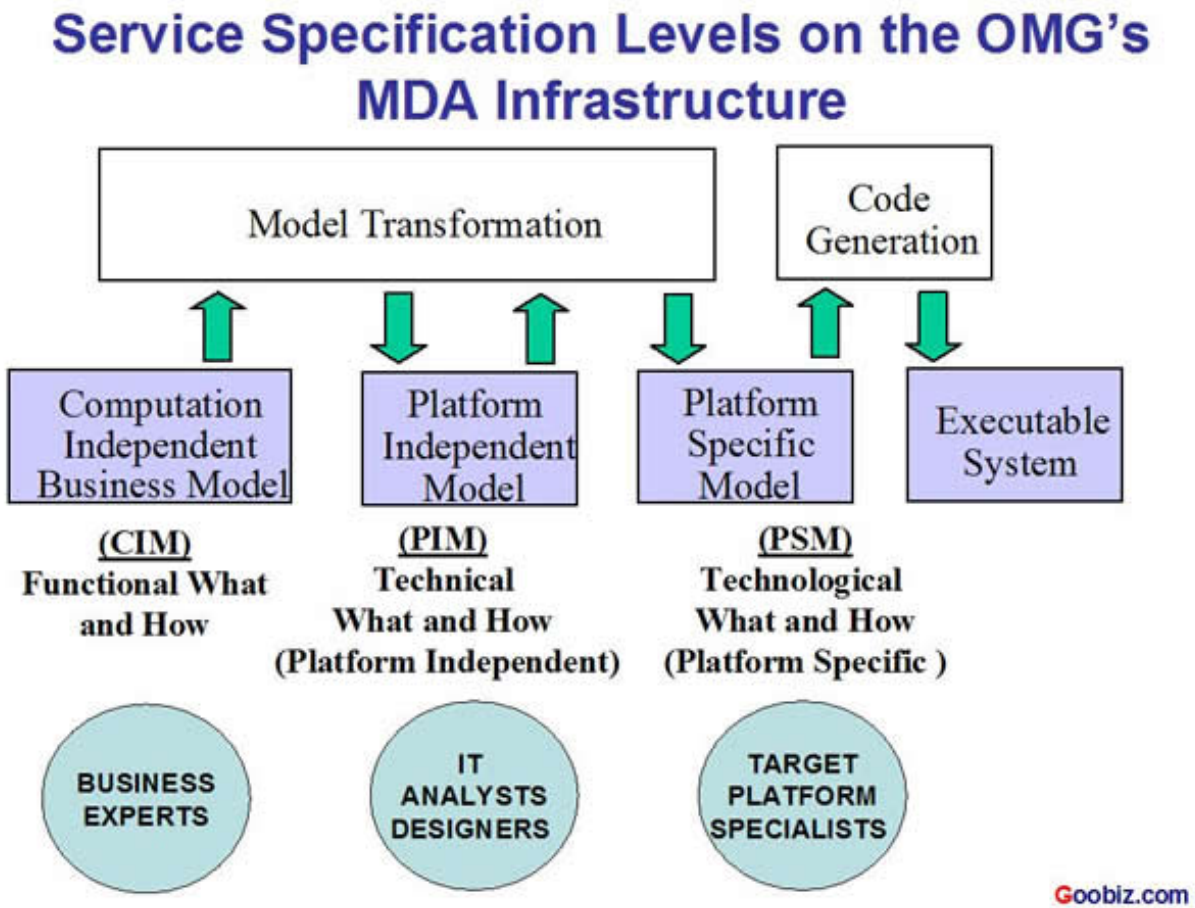
Engenharia Dirigida a Modelos (MDA)

- Que tipo de transformação é mais passível de automação: CIM->PIM ou PSM->Código?

Engenharia Dirigida a Modelos (MDA)

- A base da MDA é o MOF (Meta-Object Facility).
- MOF é o meta-modelo que serve de base para a definição de todas as linguagens de modelagem.
- A MDA também define o XMI (XML Metadata Interchange)-um formato para representar modelos em XML.
- Outro padrão da MDA é o QVT (Queries/Views/Transformations)-define uma linguagem textual e uma notação para definir consultas e transformações baseadas no MOF.
- Foco na UML.

Engenharia Dirigida a Modelos (MDA)



Abordagens MDD

- Sun Microsystems:
  - JMI (Java MetaData Interface)
  - MDR (MetaData Repository)
- Abordagem Eclipse (IBM) - EMF (Eclipse Modeling Framework), GMT (Generative Modeling Tools).

Vantagens

- Produtividade
- Portabilidade
- Interoperabilidade
- Manutenção e documentação
- Reutilização
- Verificações e otimizações
- Corretude

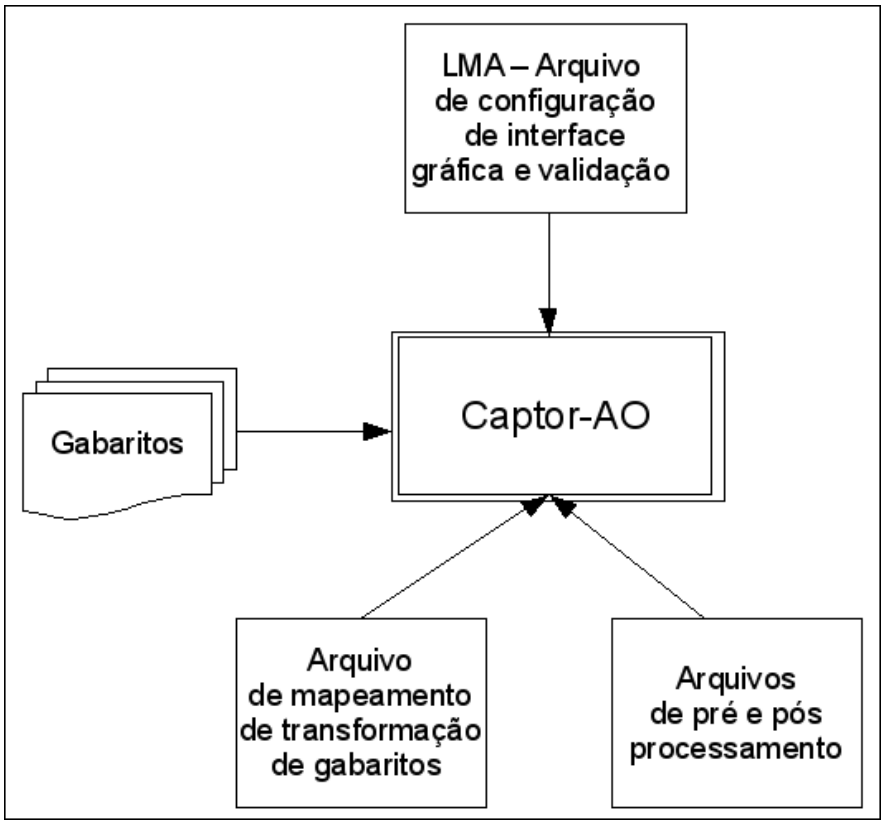
Desvantagens

- Rigidez
- Complexidade
- Desempenho
- Curva de aprendizado
- Alto investimento inicial

Geradores de Aplicações

- São utilizados em MDD como motores de transformações.
- Geradores de aplicações são ferramentas capazes de gerar artefatos a partir de uma especificação.
- Os artefatos gerados pelo gerador de aplicações podem ser segmentos de código, subrotinas, sistemas de software, testes, entre outros.
- Geradores de aplicações configuráveis (GAC) são geradores que podem ser adaptados para diferentes domínios.
- Captor - Baseado em gabaritos.

Passos Necessários



Teste de Software

- É uma forma de procurar garantir que o produto de software atende aos requisitos que foram definidos para ele. Correção, desempenho, segurança e etc.
- O que deve ser testado em um desenvolvimento baseado em modelos?

Teste de Software

- É uma forma de procurar garantir que o produto de software atende aos requisitos que foram definidos para ele. Correção, desempenho, segurança e etc.
- O que deve ser testado em um desenvolvimento baseado em modelos?
  - Ferramentas utilizadas.

Teste de Software

- É uma forma de procurar garantir que o produto de software atende aos requisitos que foram definidos para ele. Correção, desempenho, segurança e etc.
- O que deve ser testado em um desenvolvimento baseado em modelos?
  - Ferramentas utilizadas.
  - Transformações entre modelos.

Teste de Software

- É uma forma de procurar garantir que o produto de software atende aos requisitos que foram definidos para ele. Correção, desempenho, segurança e etc.
- O que deve ser testado em um desenvolvimento baseado em modelos?
  - Ferramentas utilizadas.
  - Transformações entre modelos.
  - Produto de software.

Teste de Transformação entre Modelos

- Assumimos que o motor de transformação está correto
- Assumimos que o modelo de entrada foi especificado corretamente.
- O que precisa ser testado?

Teste de Transformação entre Modelos

- Assumimos que o motor de transformação está correto
- Assumimos que o modelo de entrada foi especificado corretamente.
- O que precisa ser testado?
  - A especificação da transformação entre modelos.
- Como?

Teste de Transformação entre Modelos

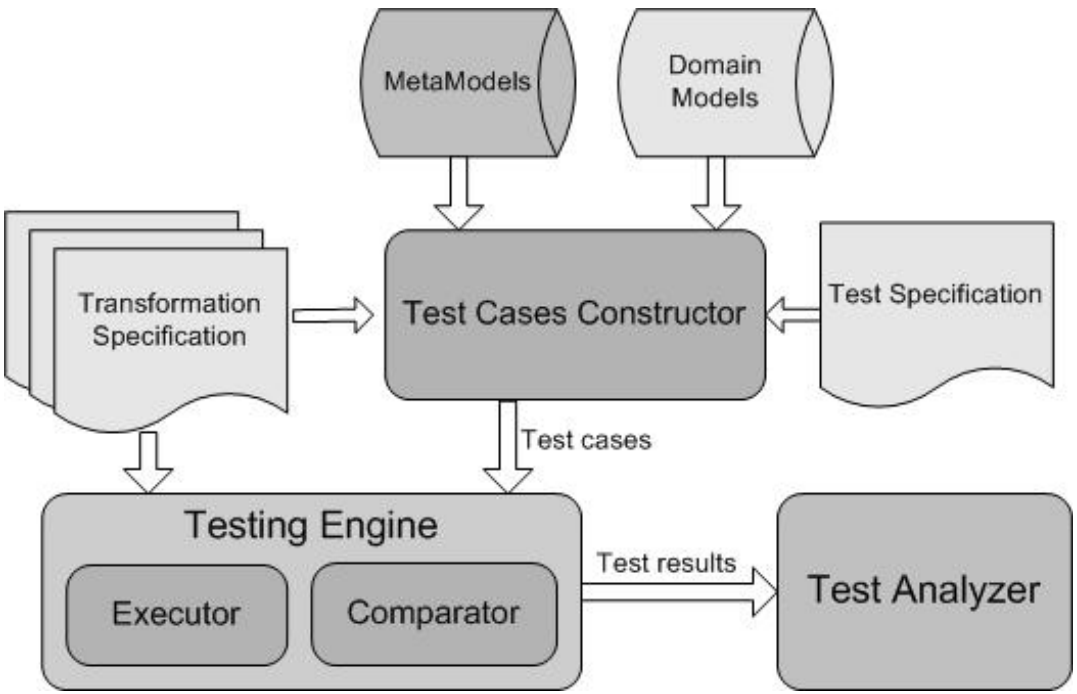
- Assumimos que o motor de transformação está correto
- Assumimos que o modelo de entrada foi especificado corretamente.
- O que precisa ser testado?
  - A especificação da transformação entre modelos.
- Como?
  - Aplicar métodos formais em uma prova de corretude.
  - Testes de execução

Teste de Transformação entre Modelos

- Vantagens dos testes de execução:
  - Relativa facilidade de execução das atividades de teste.
  - O software desenvolvido (transformação de modelos) pode ser executado no seu ambiente de execução esperado.
  - Parte do processo de teste pode ser automatizado.
- Assim como a atividade de teste convencional, os testes em especificações de modelos de transformações podem "apenas" revelar a presenta de erros e não garantir a sua ausência.

Teste de Transformação entre Modelos

- O teste de transformação de modelos envolve a execução de uma determinada especificação de transformação com os dados de teste e a comparação do resultado com a saída esperada.



Test-Code Generation in Model-Driven Systems

- Embora MDD traga muitos benefícios ao longo do desenvolvimento, precebe-se uma desvantagem durante as fases de teste e manutenção.
- Pode-se assumir que uma vez que o código é gerado automaticamente, testá-lo é redundante e dispendioso?
- Em qual contexto?
- Especificação em alto nível X Instância específica do sistema.

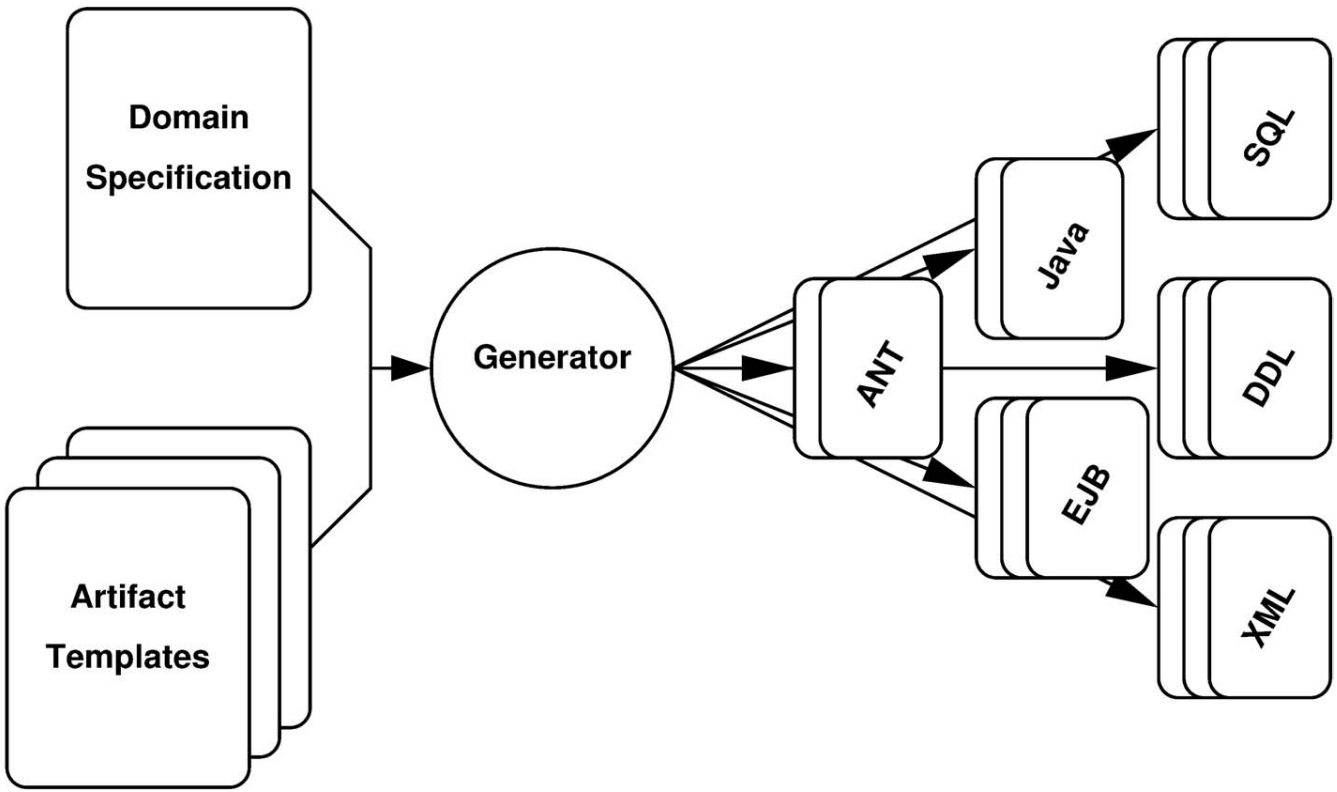
Test-Code Generation in Model-Driven Systems

- Com um apropriado nível de detalhe na especificação de componentes específicos de domínio é possível automatizar a geração de alguns ou todos os casos de teste.
- Nem todo código de um sistema feito com MDD é gerado automaticamente. Para o código que é escrito a mão deve-se fazer casos de teste de forma tradicional.
- Abordagem baseada na utilização da ferramenta MODEST (Model-Driven Enterprise System Transformer).

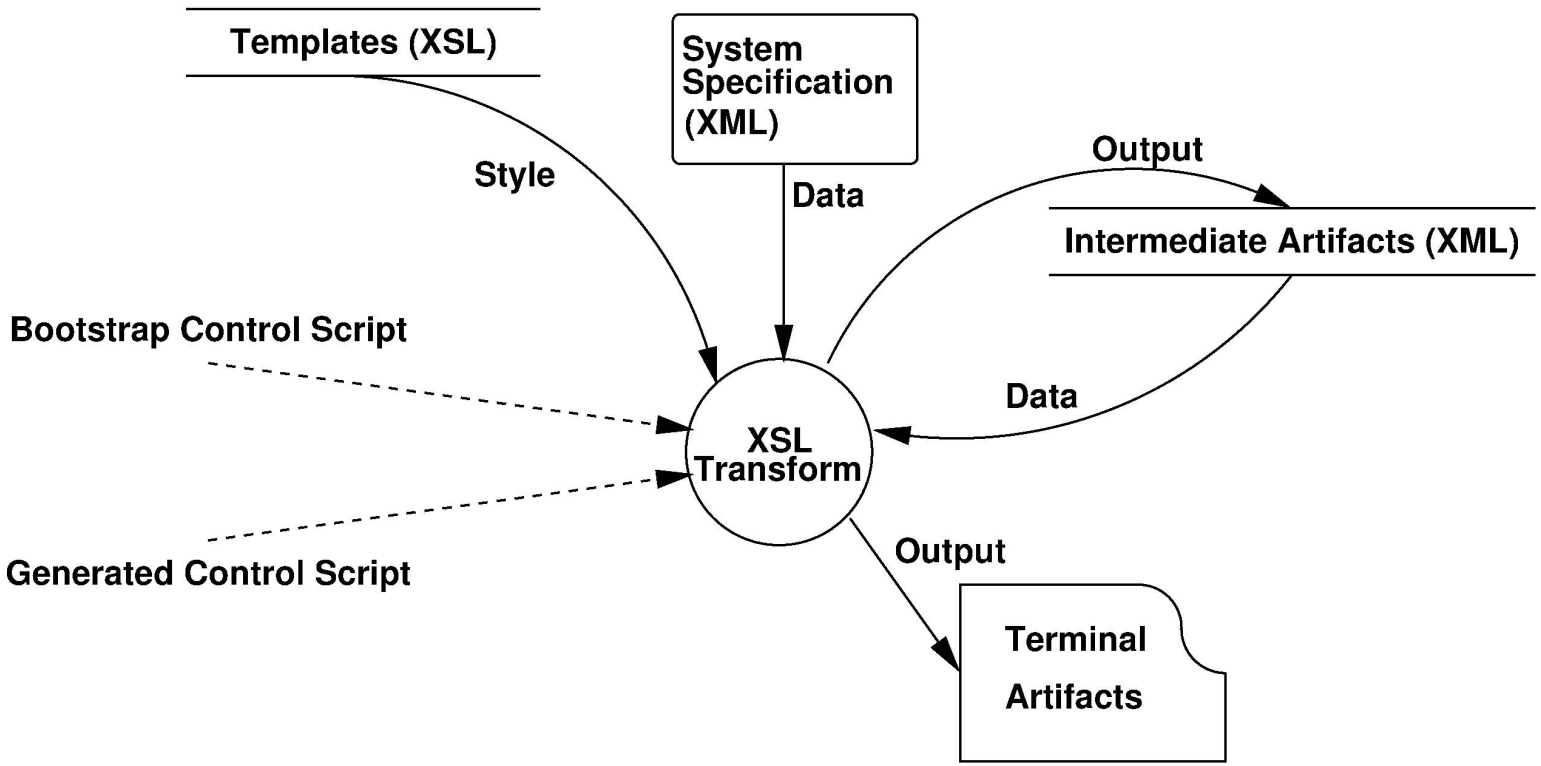
MODEST

- O desenvolvedor utiliza o MODEST na criação de um sistema específico para as necessidades de um cliente.
- Entradas do MODEST.
  - Especificação do sistema.
  - Templates de código.
- Por fim, o desenvolvedor implementa operações específicas de domínio e os casos de teste para elas.
- A geração de artefatos é feita por meio de um conjunto de transformações XSL.

MODEST


Domain
Specification
Artifact
Templates
Generator
ANT
Java
EJB
SQL
DDL
XML

MODEST


Templates (XSL)
System Specification (XML)
Style
Data
Output
Intermediate Artifacts (XML)
Bootstrap Control Script
Data
XSL Transform
Generated Control Script
Output
Terminal Artifacts

Exemplo de Especificação de Domínio

```
<domain-specification>
  <static-object name="Make">
    <attribute name="name" type="String" unique="true" required="true">
      <validator type="string-length" min="1" max="64"/>
    </attribute>
    <instance name="SUBARU" key="10">
      <attribute-value name="name">Subaru</attribute-value>
    </instance>
    <instance name="FORD" key="10">
      <attribute-value name="name">Ford</attribute-value>
    </instance></static-object>
  <managed-object name="Car" type-key="20">
    <attribute type="String" name="id" unique="true" required="true">
      <validator type="string-length" min="1" max="128"/>
      <validator type="alphanumeric-string"/></attribute>
    <attribute name="make" static-object="Make" required="true"/>
  </managed-object>
  <managed-object name="Driver" type-key="30">
    <attribute type="String" name="name" required="true">
      <validator type="string-length" min="1" max="128"/>
      <validator type="alphabetic-string"/></attribute>
    <attribute type="Integer" name="age" required="true">
      <validator type="range" min="16" max="110"/>
    </attribute></managed-object>
  <relationship src="Car" dest="Driver" type="1-n"/>
  <business-object name="IdGenerator">
    <business-method name="nextId" return="String"/>
  </business-object></domain-specification>
```

Código de teste - MODEST

- Código de teste é gerado em paralelo com o código da aplicação.
- Usado para validar e certificar atividades em tempo de desenvolvimento.
- Prover um framework para dar apoio as atividades de manutenção.
- Aspectos da geração de código de teste.
  - Geração de código para instanciar objetos gerenciados e estáticos representativos.
  - Geração de casos de teste para validar implementação de objetos estaticos.
  - Geração de casos de teste para validar implementações de objetos gerenciados.
  - Geração de framework de teste para facilitar o teste de lógica específica de domínio.

Instancias de objetos representativos - MODEST

- Todo caso de teste precisa de dados representativos para exercitar uma funcionalidade do sistema.
- São necessárias instâncias representativas dos ”objetos de domínio“ para uso no código de teste.
- Para o domínio apresentado inicialmente a classe de teste básica tem os seguintes métodos:
  - newMake() - Escolhe uma instância do Make aleatoriamente.
  - newDriver() - Popula atributos aleatoriamente com o nome base nos seus validadores.
  - newCar() - Popula o atributo id aleatoriamente usando o newMake() para pegar um Make.
- Esses algoritmos aleatórios geram valores nulos, valores em condições de fronteiras e valores no meio do intervalo de teste.

Casos de Teste para Objetos Estáticos e Gerenciados-MODEST

- Testes de unidade.
- Testes de integração

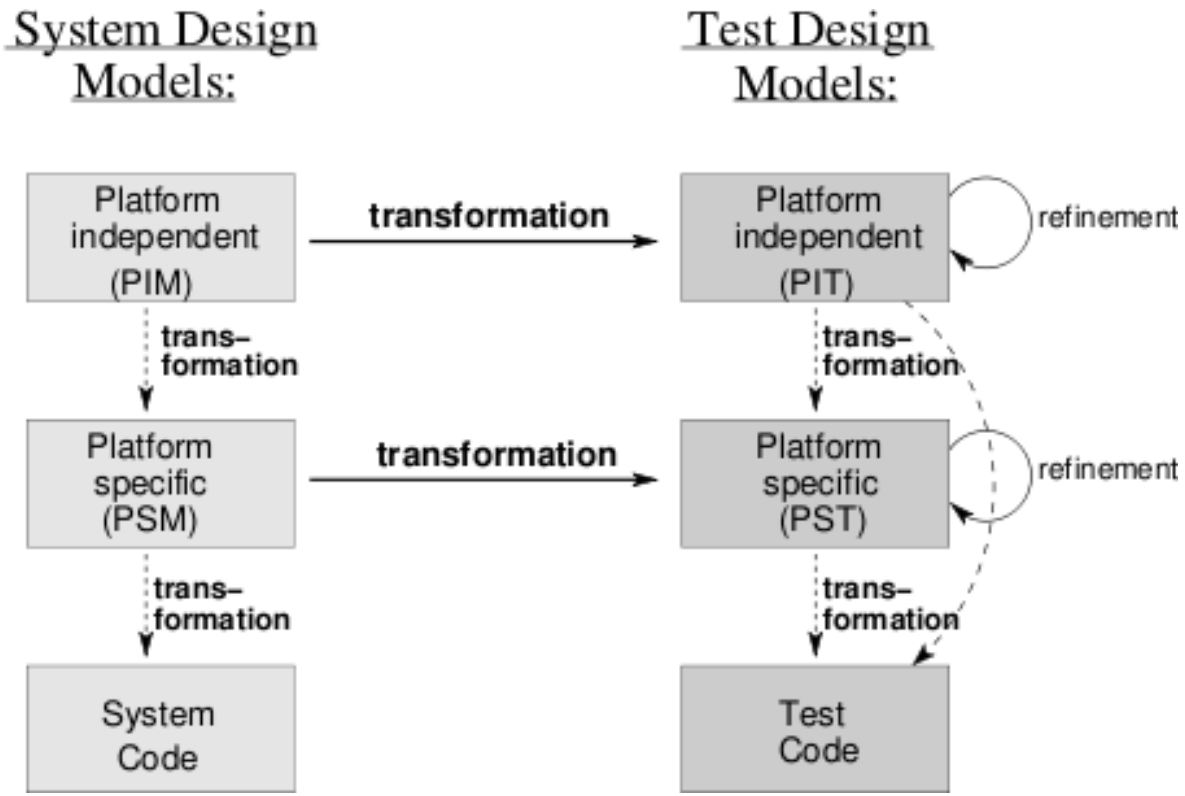
Teste dirigido por modelos (MDT)

- Geração de casos de teste a partir de modelos de acordo com um dado critério de cobertura.
- Geração de um oraculo de teste para determinar o resultado esperado de um teste.
- Execução de testes no ambiente de teste, possivelmente gerado a partir de modelos

Teste dirigido por modelos (MDT)

- Problemas:
  - Geração de casos de testes independente de plataforma e oráculo (PIT) a partir do PIM.
  - Mapeamento do PIT para plataformas alvo específicas.

UML 2.0 Testing Profile (U2TP)



UML 2.0 Testing Profile (U2TP)

- O perfil introduz 4 conceitos lógicos para definir a linguagem de modelagem de um sistema de teste:
  - Arquitetura de teste.
  - Comportamento de teste.
  - Dados de teste.
  - Tempo
- Usados para visualizar, especificar, analisar, construir e documentar.

Conceitos de Arquitetura de Teste

- Um ou mais objetos podem ser identificados como *System Under Test* (SUT).
- Componentes de teste são objetos dentro do teste de sistema que podem se comunicar com o SUT para realizar o comportamento de teste.
- Contextos de teste permitem aos usuários agrupar os casos de teste para descrever uma configuração de teste.
- Arbitragem é uma maneira de avaliar um veredito para um contexto de teste.
- O escalonador controla a execução dos testes e componentes de teste.

Conceitos de Comportamento de Teste

- O objetivo de teste define o propósito do teste.
- Diagramas de interação UML podem ser usados para definir estímulos de teste, observações, controle/invocação de testes, coordenações e ações.
- O comportamento normativo do teste é escrito em casos de teste.
- Um CT é uma operação do contexto de teste que especifica como um conjunto de componentes interage com o SUT.
- Uma ação de validação é executada por um componente de teste local para informar o veredito.
- Um veredito de teste pode ser: ”pass“, ”inconclusive“,”fail“ ou ”error“.

Conceitos de Dados de Teste

- "Wildcards" são usadas para manipular eventos inesperados.
- "Data pools" são associadas com o contexto de teste e incluem dados de teste concretos.
- Seletores de dados são operações para recuperar os dados de teste da "data pool".
- Regras de codificação permitem ao testador definir a codificação e decodificação dos dados de teste quando comunicados com o SUT.

Conceitos de Tempo

- Define conceitos de restrições e controle de comportamentos de teste com relação a tempo.
- Temporizadores são necessários para manipular e controlar o comportamento de teste e asegurar o seu fim.
- Zonas de tempo são utilizadas para agrupar componentes dentro de um sistema distribuído, permitindo a comparação de eventos de tempo na mesma zona de tempo.

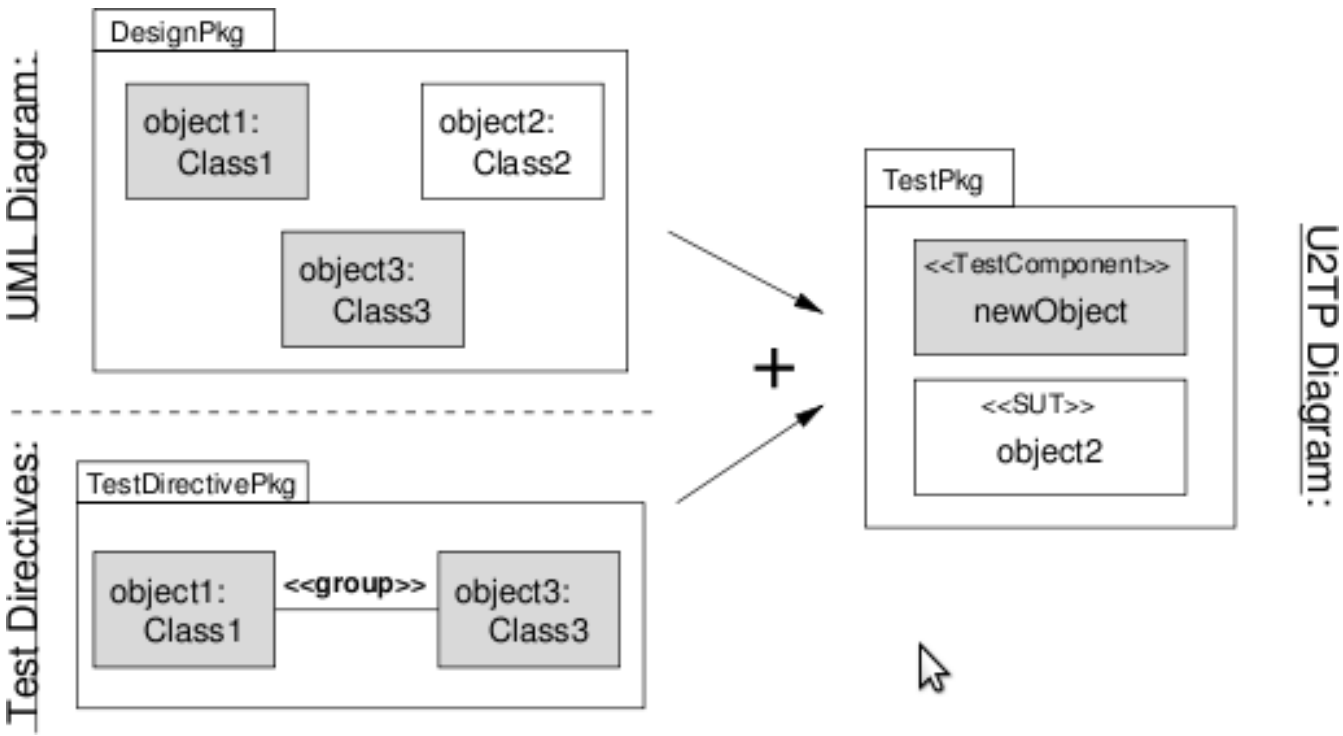
Metodo

- Definição de um novo pacote UML como o pacote de teste do sistema.
- Importe as classes e interfaces do sistema para o novo pacote.
- Inicie a partir da especificação da arquitetura de teste e depois continue com a especificação do comportamento de teste.
- Dados de teste e tempo são adicionadas nas especificações da arquitetura (Zona de tempo e "data poll") e do comportamento de teste (Temporizador ou particionamento de dados).

Metodo

- Arquitetura de teste:
  - Atribuição das classes e objetos que serão testados ao SUT.
  - Especificar uma classe de contexto de teste listando os atributos de teste e os casos de teste.
- Comportamento de teste:
  - Utilize diagramas de interação do modelo do sistema para especificar casos de teste. Um caso de teste de ser criado para cada UC.
  - Determine o veredito ao final da especificação de cada caso de teste.

UML 2.0 Testing Profile (U2TP)



Conclusões

- Pesquisas incipientes.
- Faltam detalhes sobre a geração de casos de testes e os critérios utilizados.
- No Silver Bullet - Essence and Accidents of Software Engineering.

Referências

- Czarnecki, K.; Østerbye, K.; Volter, M. Generative programming. In: ECOOP Workshops, 2002, p. 15?29.
- Lucredio, D. Uma abordagem orientada a modelos para reutilização de software. Tese de Doutoramento, Sao Carlos, SP, Brasil, director-Renata Pontin de Mattos Fortes, 2007.
- Schmidt, D. C. Model-driven engineering. IEEE Computer, v. 39, n. 2, 2006.Disponível em: http://www.truststc.org/pubs/30.html
- Shimabukuro, J. E. K. Um gerador de aplicações configurável. Dissertação de Mestrado, ICMC-USP, 2006.
- Reiko Heckel, Marc Lohmann, Towards Model-Driven Testing, Electronic Notes in Theoretical Computer Science, Volume 82, Issue 6, TACoS'03, International Workshop on Test and Analysis.

Referências

- Hailpern, B. and Tarr, P. Model-driven development: the good, the bad, and the ugly, IBM Syst. J., Volume 45.
- Yuehua, L. and Jing, Z. and Gray J. Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development
- Born M. and Schieferdecker I. and Gross H. and Santos P. Model-Driven Development and Testing ? A Case Study
- Rutherford M. and Wolf L. A. A Case for Test-Code Generation in Model-Driven Systems
- Rodrigues M. E. and Zorzo F. A. Linha de Produto de Teste Baseado em Modelos
- Zhen R. D Model-Driven Testing with UML 2.0
- Torres H. A. and Escalona J. M. and Mejías M. and Gutiérrez J. J. A MDA-BASED TESTING

Dúvidas