Arquitetura de Software
Visão Geral do Curso

Marco Túlio Valente

mtov@dcc.ufmg.br

DCC - UFMG

Parte I: Técnicas Avançadas de Modularização

Linhas de Produtos de Software

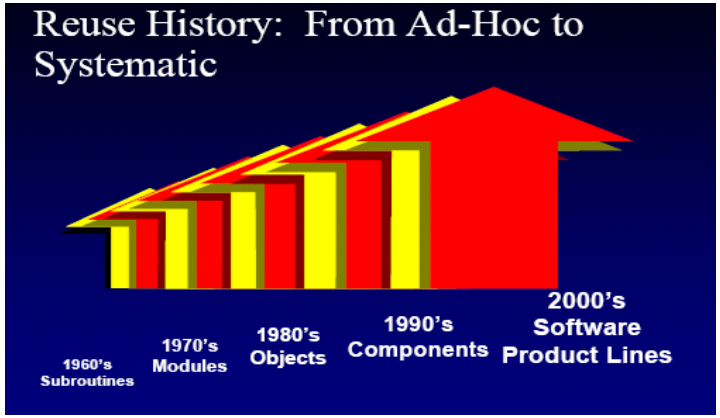
Definição

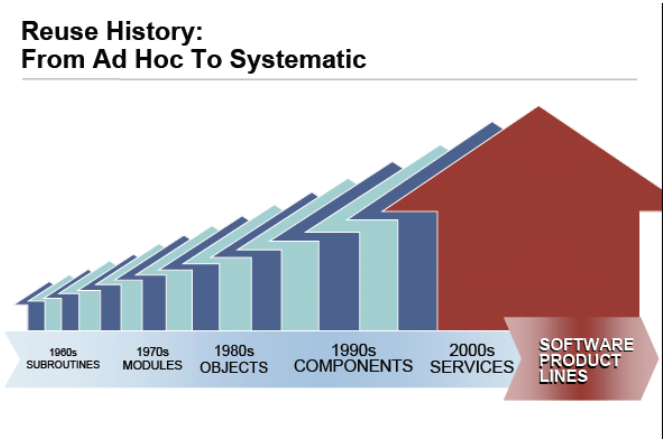
- Surgimento do termo: Workshop do SEI, 1996
  - Ou então: On the Design and Development of Program Families, David Parnas, 1976

- SEI: “A SPL is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

- Inspiração: linhas produtos industriais (customização em massa)
  - Exemplo: indústria automobilística
  - Plataforma de carro comum; a partir dessa plataforma são fabricados diversos carros; que possuem diversos itens opcionais

LPS = Reúso Sistemático

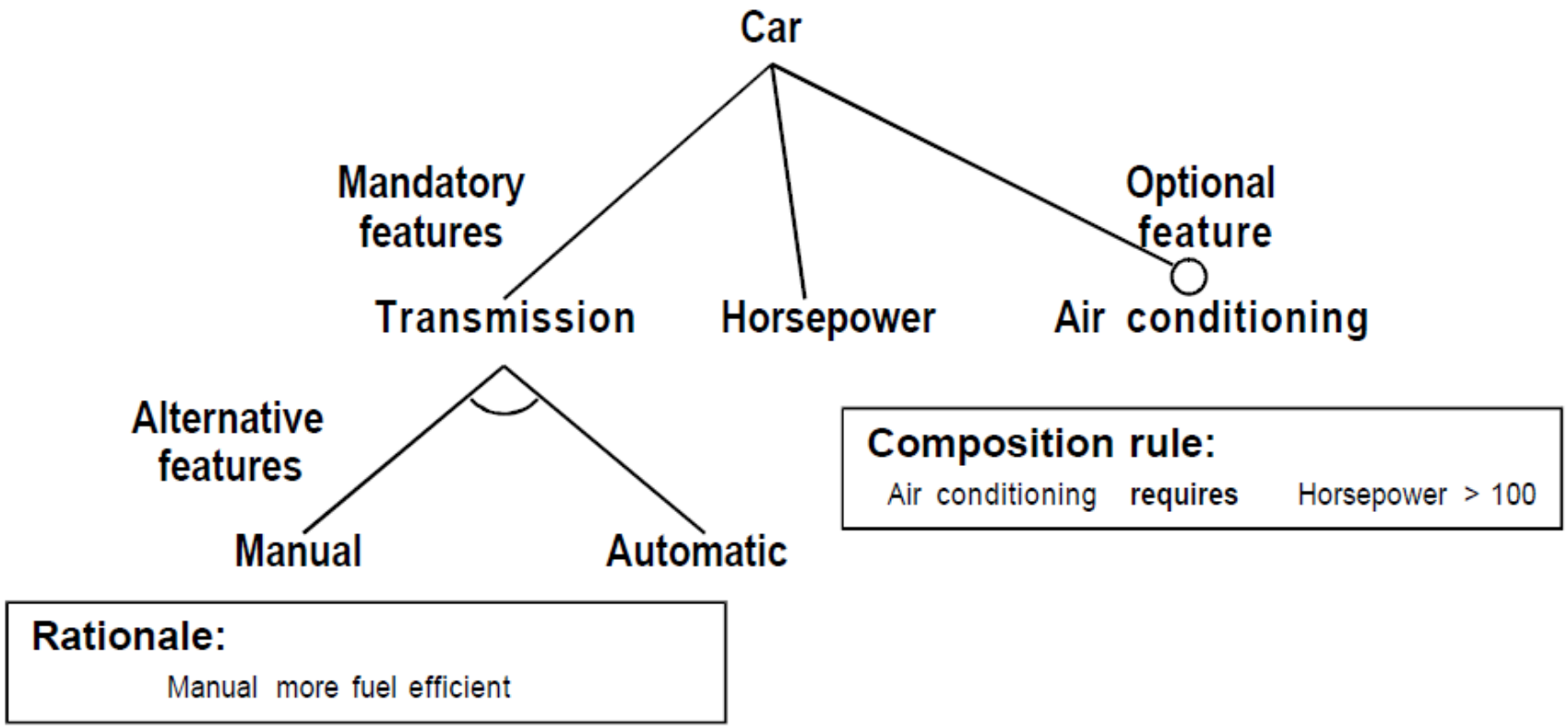




Feature Model

- Feature model: should capture the common features and differences of the applications in the domain
- Communication medium between users and developers.
- To the users:
  - Shows what the standard features are
  - What other features they can choose
  - When they can choose them
- To the developer:
  - Indicates what needs to be parameterized

Feature Model



SPL Extraction

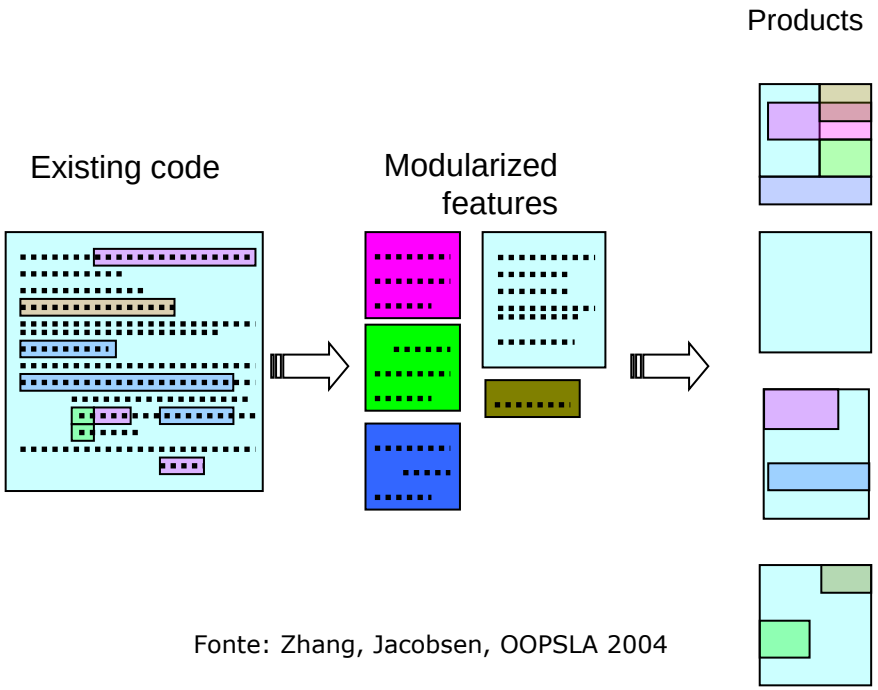
Motivation

- SPL extraction is a time-consuming task

- Two approaches for extracting SPL:
  - Compositional-based
  - Annotation-based

Compositional-based Approaches

- Aspects (AspectJ)



Fonte: Zhang, Jacobsen, OOPSLA 2004

- Physical “modularization”
- Slow adoption

Annotation-based Approaches

- Example: preprocessors

```
boolean push(Object o) {
  Lock lk = new Lock();
  if (lk.lock() == null) {
    Log.log("lock failed");
    return false;
  }
  elements[top++]= o;
  size++;
  lk.unlock();
  if ((size % 10) == 0)
    snapshot("db");
  if ((size % 100) == 0)
    replicate("db","srv2");
  return true;
}
```

```
boolean push(Object o) {
  #ifdef MULTITHREADING
  Lock lk = new Lock();
  if (lk.lock() == null) {
    #ifdef LOGGING
    Log.log("lock failed");
    #endif
    return false;
  }
  #endif
  elements[top++]= o;
  size++;
  #ifdef MULTITHREADING
  lk.unlock();
  #endif
  #ifdef SNAPSHOT
  if ((size % 10) == 0)
    snapshot("db");
  #endif
  #ifdef REPLICATION
  if ((size % 100) == 0)
    replicate("db","srv2");
  #endif
  return true;
}
```

- Widely adopted
- Problems: annotation hell; code pollution

Visual Annotations

- CIDE: Colored IDE (Eclipse + background colors)

```
boolean push(Object o) {
  Lock lk = new Lock();
  if (lk.lock() == null) {
    Log.log("lock failed");
    return false;
  }
  elements[top++]= o;
  size++;
  lk.unlock();
  if ((size % 10) == 0)
    snapshot("db");
  if ((size % 100) == 0)
    replicate("db","srv2");
  return true;
}
```

```
boolean push(Object o) {
  Lock lk = new Lock();
  if (lk.lock() == null) {
    Log.log("lock failed");
    return false;
  }
  elements[top++]= o;
  size++;
  lk.unlock();
  if ((size % 10) == 0)
    snapshot("db");
  if ((size % 100) == 0)
    replicate("db","srv2");
  return true;
}
```

- Less code polution than #ifdefs
- Problem: colors assigned manually (repetitive, error-prone etc)

Sugestões de Projetos

1. Realizar um *survey* sobre principais linhas de produtos de software usadas em papers da área
   - Objetivo: criar um “repositório” de LPS, para facilitar pesquisas na área
   - Tarefas:
     - “Instalar/usar/configurar” linhas de produtos
     - Descobrir ou prover documentação (incluindo *feature model*)
     - Elaborar artigo em inglês
     - Disponibilizar página web

2. Modelar variabilidades de um sistema não-trivial:
   - Criar *feature model*
   - Classificar *features* (alternativas, opcionais etc)
   - Definir restrições entre *features*
   - Sugestões de linguagens: TVL, CDL e Kconfig

Sugestões de Projetos

3. Avaliação da linguagem Clafer para modelagem de SPLs + comparação com outras linguagens (TVL, CDL, Kconfig)
   - Criar (ou reusar) uma LPS de exemplo
   - Modelar em Clafer
   - Avaliar a linguagem
   - Comparar com outras linguagens (TVL, CDL, Kconfig)

4. Extrair *features* de um sistema não-trivial
   - Exemplo: ArgoUML-SPL (http://argouml-spl.tigris.org/)

Parte II – Recuperação e Conformação Arquitetural

Recuperação Arquitetural

Dependency Structure Matrix (DSM)

- Invented for optimizing product development processes

- Design Rules: The Power of Modularity, Carliss Y. Baldwin and Kim B. Clark, 2000
  - Industry has experienced unimaginable levels of innovation and growth because it embraced the concept of modularity, building complex products from smaller subsystems that can be designed independently yet function together as a whole

Dependency Structure Matrix (DSM)

- The matrix is a simple adjacency matrix with:
  - Tasks labeling the horizontal and vertical axes
  - Mark in the i^{th} column, j^{th} row: the i^{th} task depends on the j^{th}
- Figure 1 shows a simple DSM:
  - Column 1: task A depends on task C
  - Column 3: task C depends on tasks A and B

		1	2	3	4
Task A	1	.		X	X
Task B	2		.	X	
Task C	3	X		.	X
Task D	4				.

Figure 1: A Simple DSM

Architectural Patterns

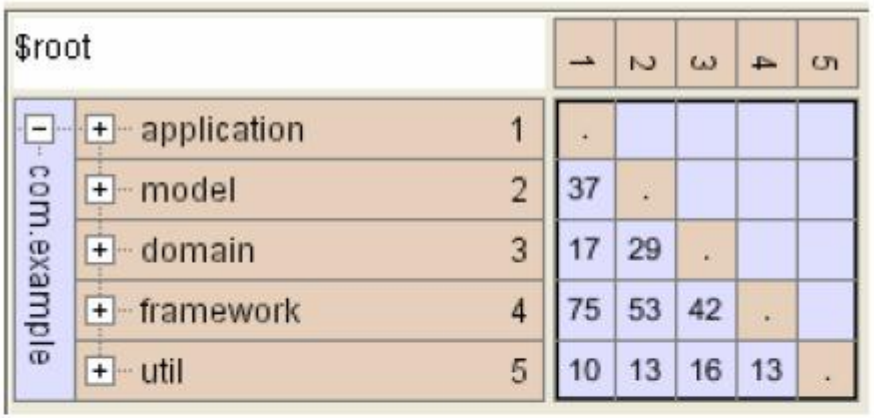


Figure 5: Layered System

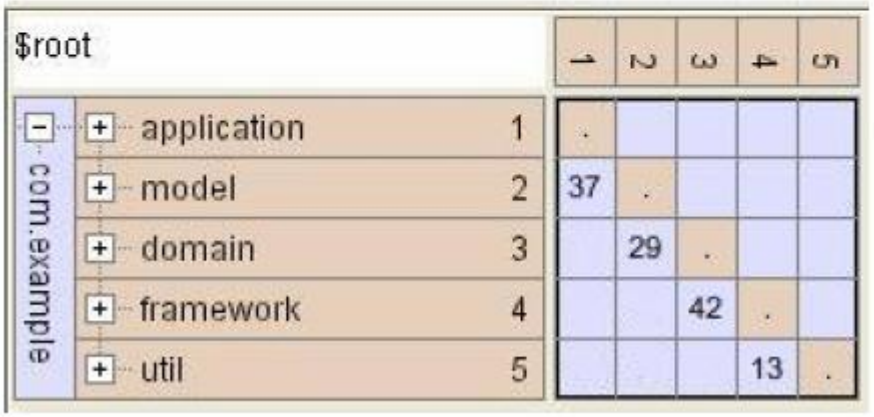
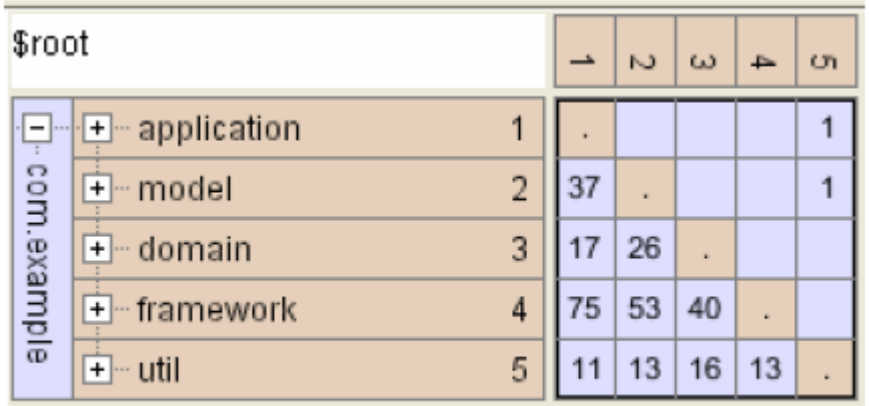


Figure 6: Strictly Layered System

Imperfectly Layered System

- Module *util* depends on *application* and *model*
- Dependency strengths suggests that this dependency is not as strong as the reverse dependency

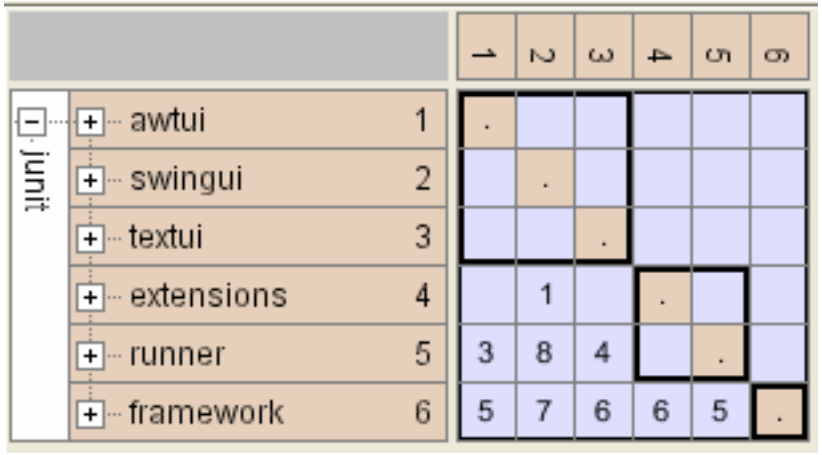


$root			1	2	3	4	5
com.example	application	1	.				1
	model	2	37	.			1
	domain	3	17	26	.		
	framework	4	75	53	40	.	
	util	5	11	13	16	13	.

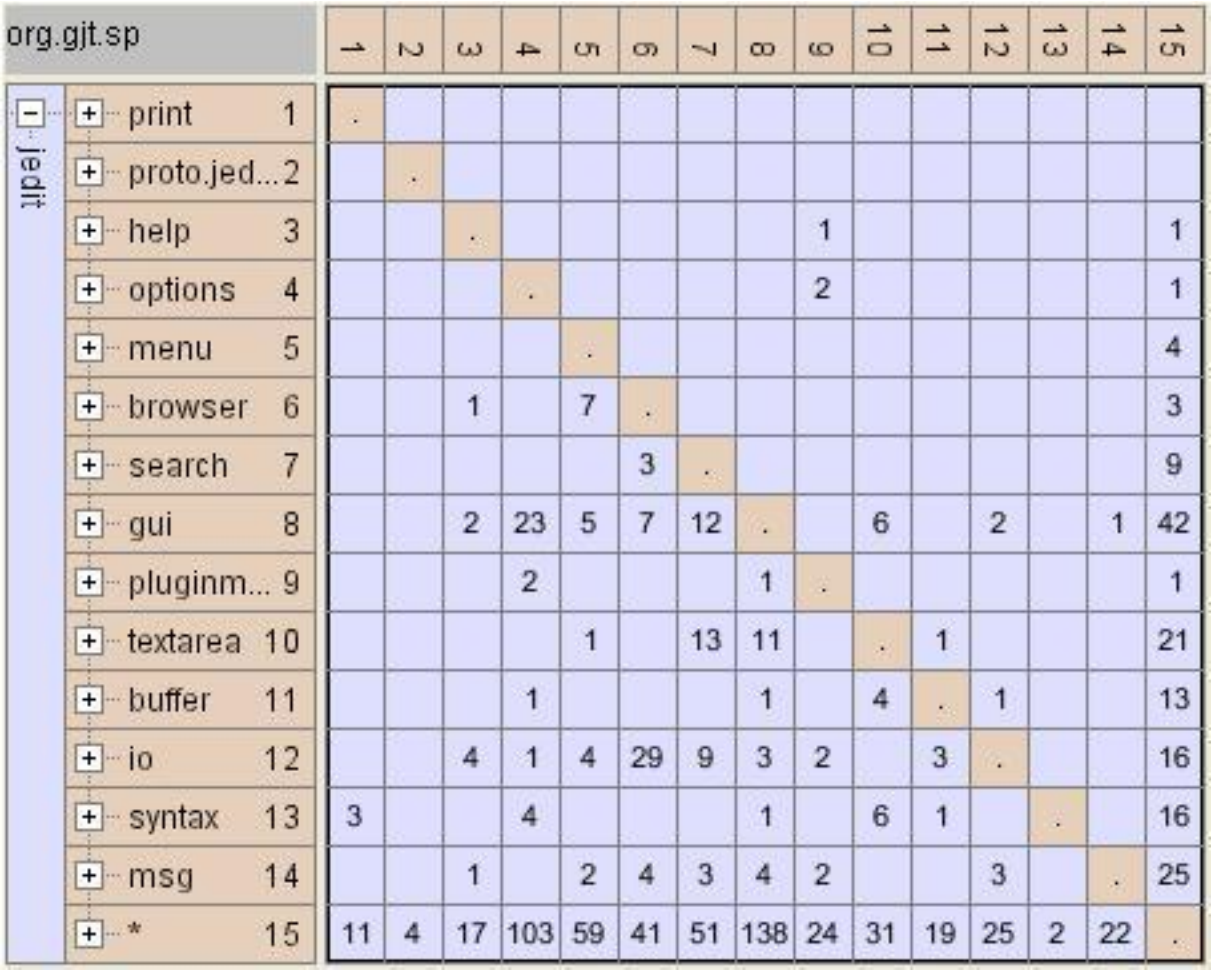
Figure 9: Imperfectly Layered System

DSM for JUnit

- Shows that *JUnit* is a layered system with clean separation of the user interface layers from the underlying core logic.



DSM for JEdit



org.gjt.sp		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
jedit	print 1	.														
	proto.jed... 2		.													
	help 3			.						1						1
	options 4				.					2						1
	menu 5					.										4
	browser 6			1		7	.									3
	search 7						3	.								9
	gui 8			2	23	5	7	12	.		6		2		1	42
	pluginm... 9				2				1	.						1
	textarea 10					1		13	11		.	1				21
	buffer 11				1				1		4	.	1			13
	io 12			4	1	4	29	9	3	2		3	.			16
	syntax 13	3			4				1		6	1		.		16
	msg 14			1		2	4	3	4	2			3		.	25
	* 15	11	4	17	103	59	41	51	138	24	31	19	25	2	22	.

Conformação Arquitetural

Problema

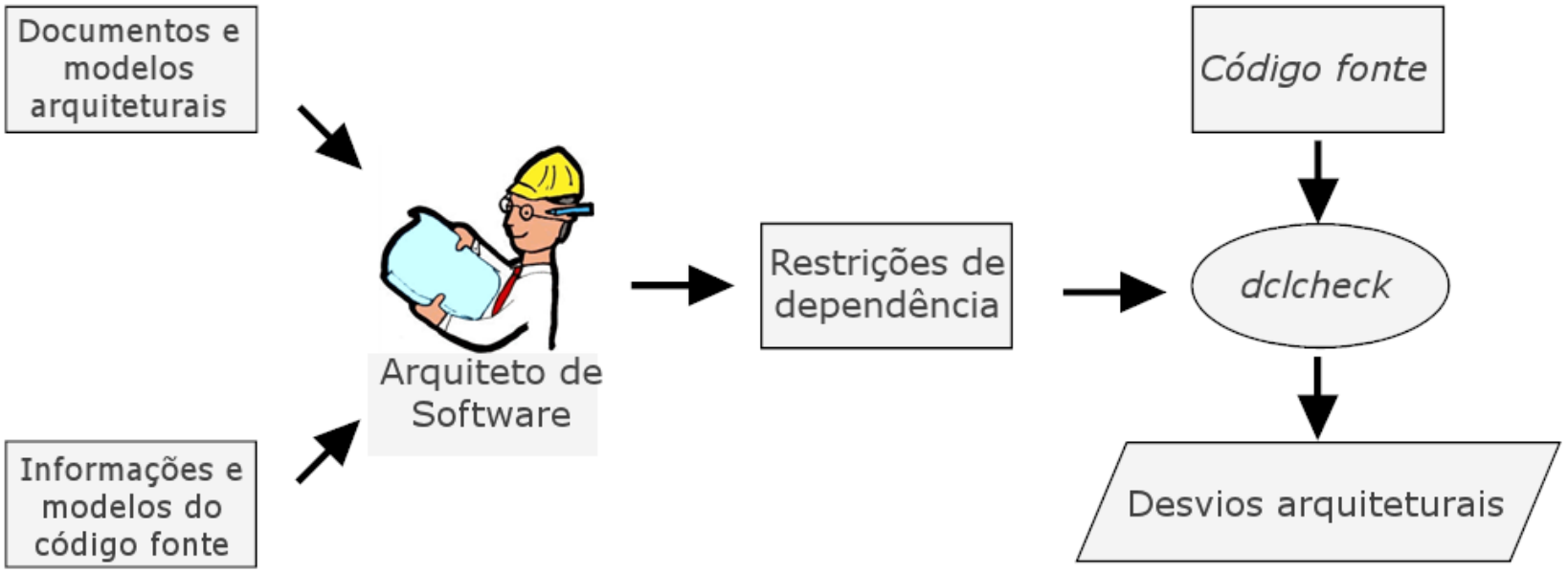
- A arquitetura [development view] de um sistema define:
  - Como ele é estruturado em componentes
  - Restrições sobre como tais componentes devem interagir
- A arquitetura documentada de um sistema – se disponível – geralmente não reflete a sua implementação real
- Desvios arquiteturais são comuns
  - Geralmente, não são capturados e resolvidos
  - Levam a fenômenos conhecidos como erosão e desvio arquitetural
- O processo de erosão arquitetural faz com que os benefícios proporcionados por um projeto arquitetural sejam anulados

Uma Solução

- Conformação arquitetural usando análise estática
- Hipótese: dependências inter-modulares impróprias são uma fonte importante de violações arquiteturais
  - Contribuem para o processo de erosão arquitetural
- Exemplo
  - Sistema organizado nas camadas M_p, M_{p-1}, ... , M_0
- Linguagens de programação não proveem meios para restringir dependências inter-modulares
  - Modificadores de visibilidade (*public, private,* etc) não são suficientes

DCL

- Uma linguagem de domínio específico para restrição de dependência
- Permite que arquitetos definam dependências aceitáveis e inaceitáveis de acordo com a arquitetura planejada



Violações Arquiteturais

- Como resultado da verificação de uma restrição, os seguintes tipos de violação podem ser detectados:

- **Divergência**
  - Quando uma dependência existente no código fonte viola uma restrição de dependência especificada

- **Ausência**
  - Quando o código fonte não possui uma dependência que deveria existir de acordo com uma restrição de dependência especificada

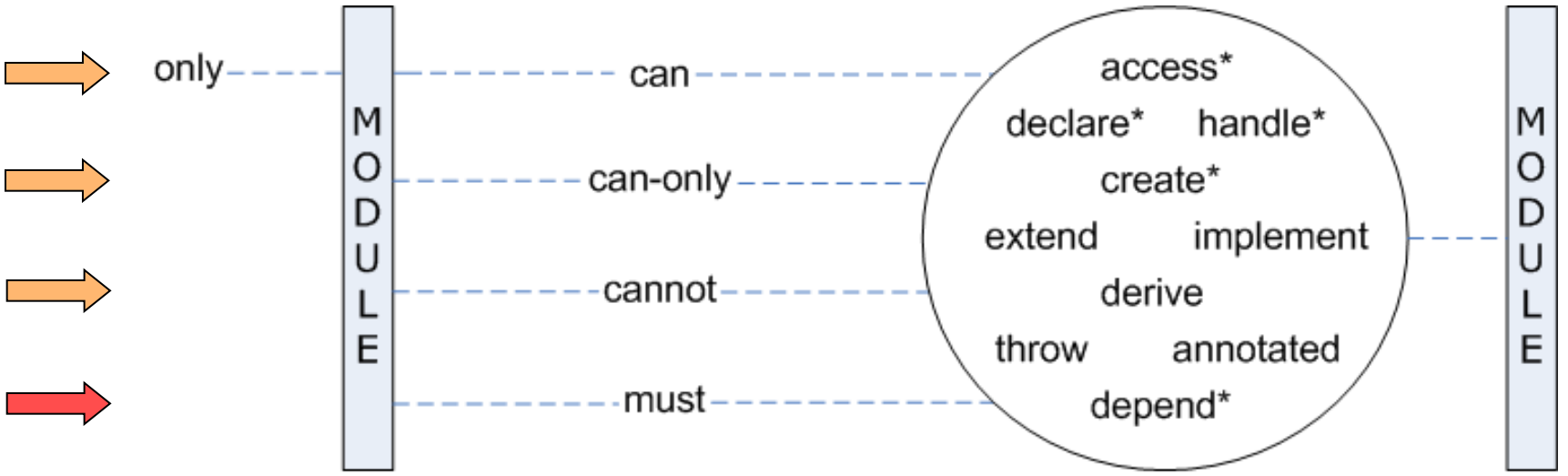
Linguagem DCL

- Linguagem de domínio específico, declarativa e estaticamente verificável
- Objetivo: Permitir a definição de restrições estruturais entre módulos
- Definição de módulos:

```
module View: org.foo.view.*
module Model: org.foo.model.**
module Remote: java.rmi.UnicastRemoteObject+
module Frame: "org.foo.[a-zA-Z0-9/.]*Frame"
```

Linguagem DCL

- Resumo da sintaxe DCL para declaração de restrições de dependência para capturar **divergências** e **ausências**:



* Restrição *must* não disponível

Exemplo

```
module View: myapp.view.View+
module JavaAwtSwing: java.awt.**,
                     javax.swing.**

only View can-depend JavaAwtSwing
```

- Sintaxe da linguagem DCL próxima a linguagem utilizada pelos arquitetos

- Mais informações:
  - Ricardo Terra; Marco Tulio Valente. A Dependency Constraint Language to Manage Object-Oriented Software Architectures. Software: Practice and Experience, 2009.

Ferramenta **`dclcheck`**

- Verifica se o código fonte respeita restrições de dependência definidas em DCL
- Implementada como *plug-in* para a IDE Eclipse
- Disponível em: **`http://www.dcc.ufmg.br/~mtov/dcl`**

Sugestões de Projetos

1. Implementação de ferramenta para extração de DSMs (para Java ou outra linguagem) + Exemplo

2. Aplicação da linguagem DCL para detectar violações em um sistema Java não-trivial (idealmente de código aberto)

3. Implementação de DCL para outras linguagens estaticamente tipadas (exemplo: C# ou C++) + Avaliação

4. Implementação de DCL para linguagens dinamicamente tipadas (exemplo: Ruby ou Python; possivelmente como uma “internal DSL”) + Avaliação

Sugestões de Projetos

5. Implementação de DCL para sistemas em Scala (provavelmente, como uma "internal DSL" também) + Avaliação

6. Extensão de DCL para componentes de granularidade mais fina. Exemplos: constituir módulos como "classes anotadas com X", "classes que dependem de Swing", etc. + Avaliação

7. Implementação de uma nova linguagem/solução para conformação arquitetural, possivelmente para um domínio e/ou arquitetura específica (exemplo: sistemas web) (ambicioso!)

8. Em um sistema legado com problemas de erosão arquitetural, analisar e caracterizar as refatorações necessárias para reverter esse processo (a exemplo do paper sobre remodularização do sistema bancário)

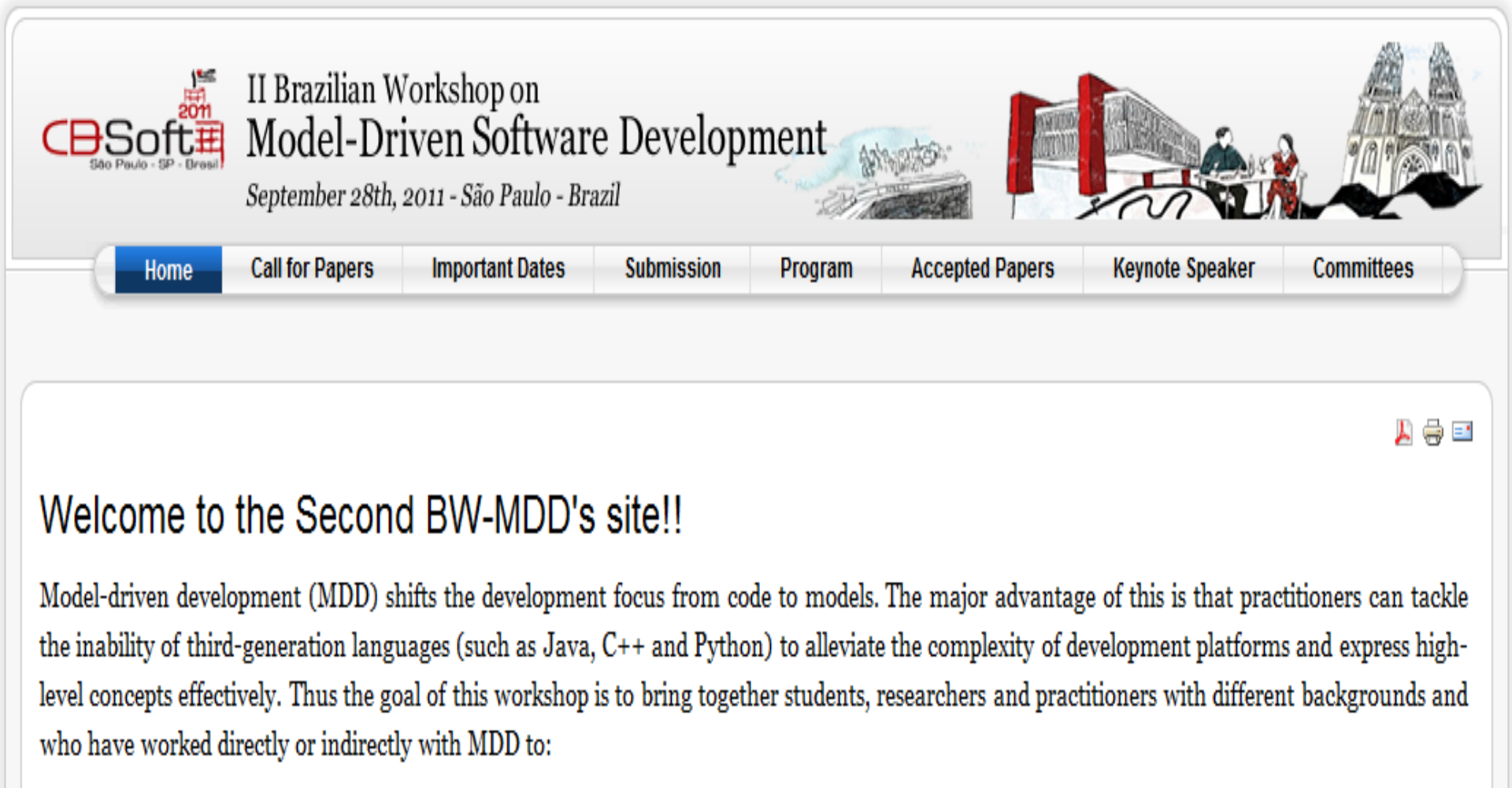
Parte III: MDA & DSL

Model-Driven Development

Model-Driven Development

- MDD shifts the development focus from code to models.
- MDD faces the substantial challenge of carrying models through to implementation
- MDD can help software development teams concentrate their efforts on modeling and generating much of the code needed

Principal Problema: “Overselling”



CFP é mais pé-no-chão

- “O tema central do II WB-DSDM é a utilização de modelos dentro do processo produtivo de software.

- Entendemos por processo produtivo de software um conjunto de atividades inerentes à construção e à manutenção de um sistema de software.

- Neste contexto, modelos podem ser utilizados para, por exemplo, expressar requisitos, especificar testes, verificar propriedades dos modelos e gerar parcial ou integralmente código fonte”

Steimann's verdict on MDD

- Models have their greatest value for people who cannot program.
- Unfortunately, this is not because models are more expressive than programs (in the sense that they are capable of expressing complex things in a simple manner) but because they oversimplify.
- To make a useful program (i.e., one that meets its users' expectations) from a model, the model must be so complicated that people who cannot program cannot understand it, and people who can program would rather write a program than draw the corresponding model.

Steimann's verdict on MDD

- Even though (over) simplification can be useful sometimes, it is certainly not a sufficient basis for creating a satisfactory end product

- From: http://queue.acm.org/detail.cfm?id=1113336

Linguagens Específicas de Domínio

Linguagens Específicas de Domínio

- Linguagens para realizar uma tarefa específica:
  - Procurar por uma string em um arquivo texto (grep, sed, awt)
  - Realizar uma consulta em um banco de dados (SQL)
  - Especificar um analisador léxico (Lex)
  - Especificar um analisador sintático (Yacc)
  - Desenhar um grafo (Graphviz)
  - Especificar dependências de compilação (makefile)
  - Especificar o layout de uma página Web (CSS)

- Objetivo: simplificar programação de tarefas específicas

- Linguagens de Próposito Geral vs Linguagens Específicas de Domínio

- DSL não são um conceito novo!
- Tendência recente: incentivar o uso de DSLs em vários sistemas

Domain-Specific Languages

Martin Fowler,
Addison-Wesley, 2011

Exemplo: Gothic Security

- Empresa que vende sistemas de segurança.
- Principal sistema:
  - Permite ter um "compartimento secreto" em casas
  - Compartimento é aberto após uma sequência de eventos

- Exemplo (Miss Grant): compartimento é aberto quando ela:
  - fecha a porta do quarto
  - abre uma gaveta
  - liga uma luz

- Sistema inclui: sensores (que enviam mensagens) e controlador

Exemplo: Gothic Security

- Clientes podem "configurar" seus compartimentos secretos
- Família de sistemas:
  - Reúsa componentes e comportamentos
  - Mas com diferenças importantes
- Objetivo:
  - Projetar esse sistema de forma a facilitar sua instalação

Miss Grant's Controller

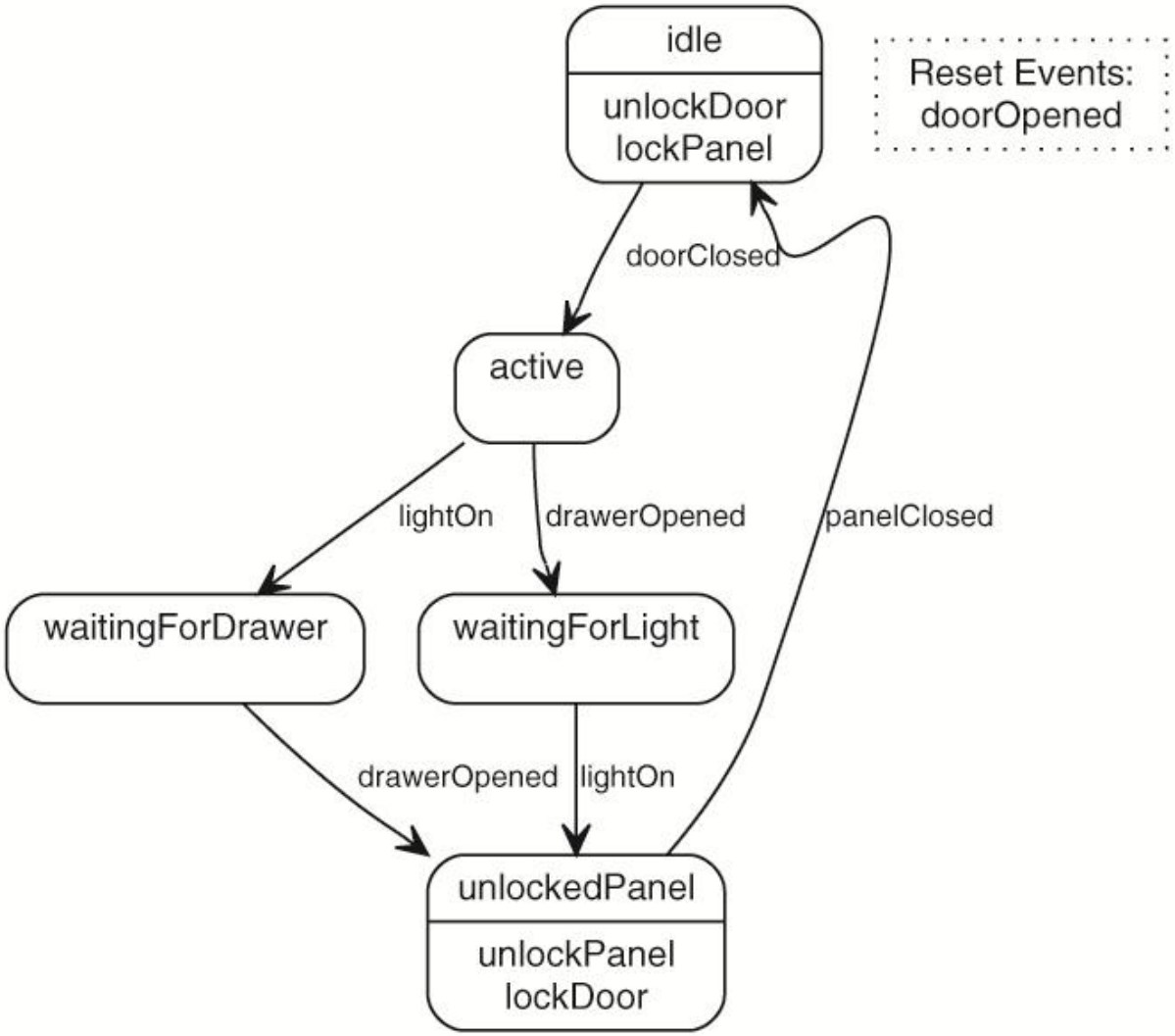
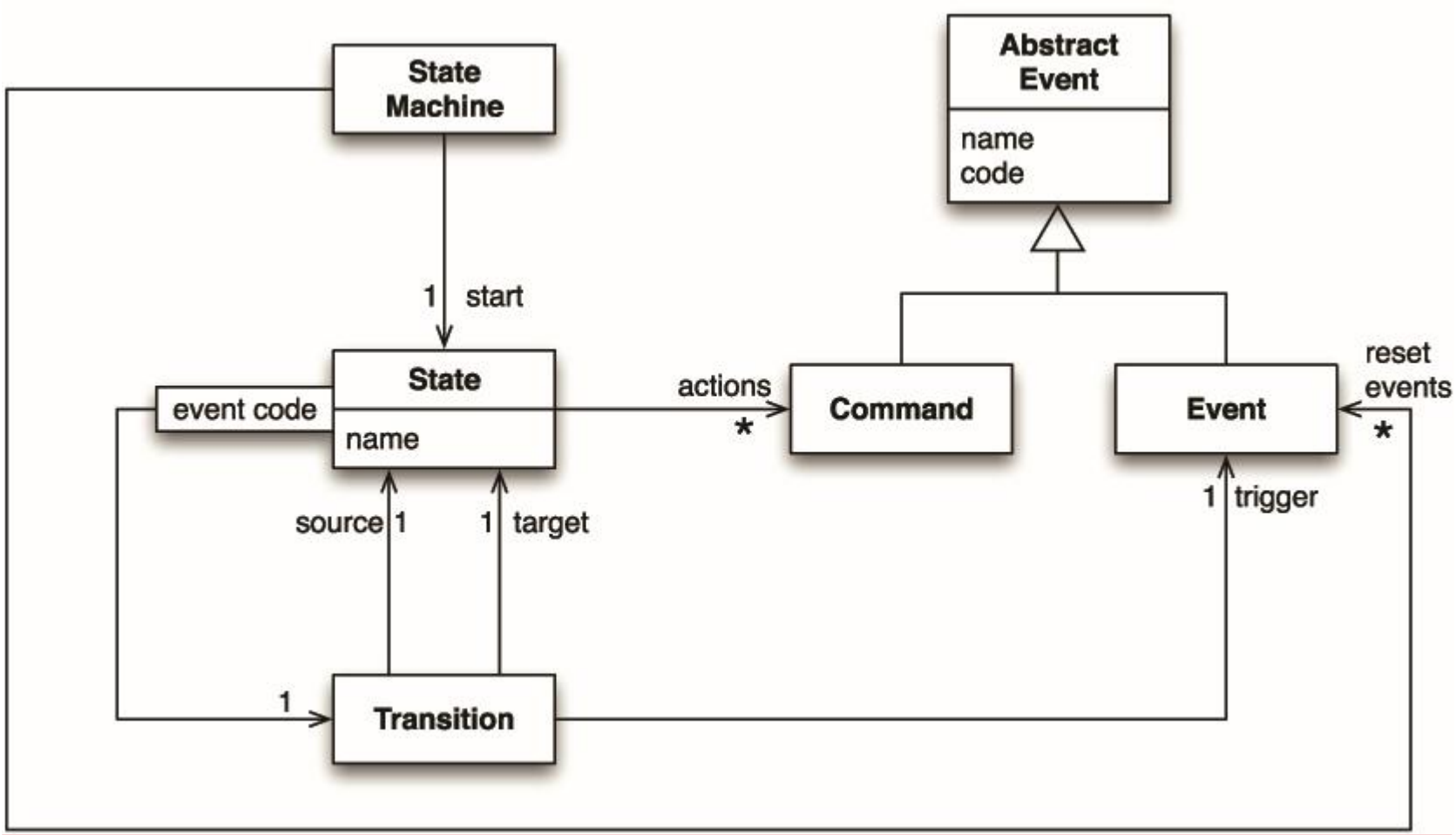


Diagrama de Classes



Programming Miss Grant's Controller

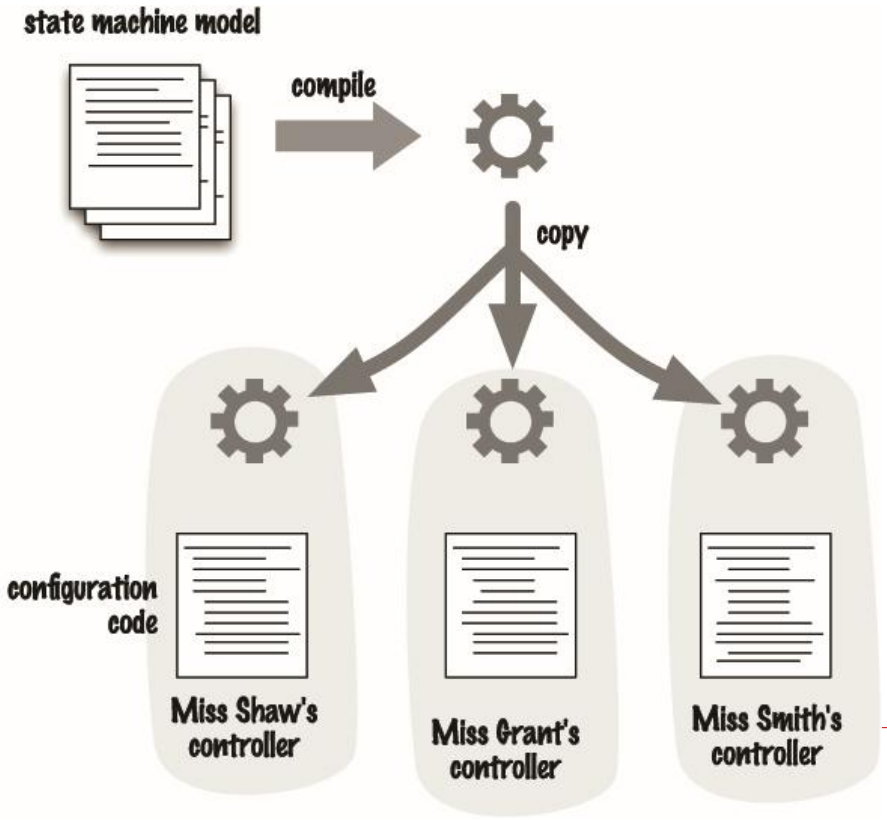
```
Event doorClosed = new Event("doorClosed", "D1CL");
Event drawerOpened = new Event("drawerOpened", "D2OP");
Event lightOn = new Event("lightOn", "L1ON");
Event doorOpened = new Event("doorOpened", "D1OP");
Event panelClosed = new Event("panelClosed", "PNCL");

Command unlockPanelCmd = new Command("unlockPanel", "PNUL");
Command lockPanelCmd = new Command("lockPanel", "PNLK");
Command lockDoorCmd = new Command("lockDoor", "D1LK");
Command unlockDoorCmd = new Command("unlockDoor", "D1UL");

State idle = new State("idle");
State activeState = new State("active");
State waitingForLightState = new State("waitingForLight");
State waitingForDrawerState = new State("waitingForDrawer");
State unlockedPanelState = new State("unlockedPanel");

StateMachine machine = new StateMachine(idle);
```

Programming Miss Grant's Controller

```
idle.addTransition(doorClosed, activeState);
 idle.addAction(unlockDoorCmd);
 idle.addAction(lockPanelCmd);

activeState.addTransition(drawerOpened,
 waitingForLightState);
 activeState.addTransition(lightOn,
 waitingForDrawerState);

waitingForLightState.addTransition(lightOn,
    unlockedPanelState);

waitingForDrawerState.addTransition(drawerOpened,
 unlockedPanelState);

unlockedPanelState.addAction(unlockPanelCmd);
 unlockedPanelState.addAction(lockDoorCmd);
 unlockedPanelState.addTransition(panelClosed,idle);

machine.addResetEvents(doorOpened);
```

Single library + multiple configurations

- Projeto permite separação entre:
  - Código comum (máquina de estados)
  - Código de configuração



Alternativa I: Configuração XML

```
<stateMachine start = "idle">
 <event name="doorClosed" code="D1CL"/>
 <event name="drawerOpened" code="D2OP"/>
 <event name="lightOn" code="L1ON"/>
 <event name="doorOpened" code="D1OP"/>
 <event name="panelClosed" code="PNCL"/>

 <command name="unlockPanel" code="PNUL"/>
 <command name="lockPanel" code="PNLK"/>
 <command name="lockDoor" code="D1LK"/>
 <command name="unlockDoor" code="D1UL"/>

 <state name="idle">
 <transition event="doorClosed" target="active"/>
 <action command="unlockDoor"/>
 <action command="lockPanel"/>

 </state>
 <state name="active">
 <transition event="drawerOpened" target="waitingForLight"/>
 <transition event="lightOn" target="waitingForDrawer"/>
```

Alternativa I: Configuração XML

- Vantagem:
  - Não precisa ser compilado.
  - Novas instalações não requerem distribuição de novo JAR
  - “Cada vez, programamos mais em XML e menos em LPs”

- Desvantagem: sintaxe

Alternativa II: DSL

```
events
 doorClosed D1CL
 drawerOpened D2OP
 lightOn L1ON
 doorOpened D1OP
 panelClosed PNCL end

resetEvents
 doorOpened
end

commands
 unlockPanel PNUL
 lockPanel PNLK
 lockDoor D1LK
 unlockDoor D1UL
 end
```

```
state idle
 actions {unlockDoor lockPanel}
 doorClosed => active
end

state active
 drawerOpened => waitingForLight
 lightOn => waitingForDrawer
end

state waitingForLight
 lightOn => unlockedPanel
end

state waitingForDrawer
 drawerOpened => unlockedPanel
end

state unlockedPanel
 actions {unlockPanel lockDoor}
 panelClosed => idle
end
```

DSL Externas e Internas

- Externas: DSL com uma sintaxe própria
- Internas: DSL que “pega carona” na sintaxe de uma LP

Exemplo de DSL Interna

```
Order o = new Order();
Product p1 = new Product(1,Product.find(“Billy”));
o.addProduct(p1);
Product p2 = new Product(2,Product.find(”Janso"));
o.addProduct(p2);
Product p3 = new Product(4,Product.find(“Traby"));
o.addProduct(p3);
o.setPriorityRush(true);
customer.addOrder(o);
```

Exemplo de DSL Interna

```
customer.newOrder()
        .with(1, "Billy")
        .with(2, "Janso")
        .with(4, "Traby")
        .priorityRush()
        .done();
```

Exemplo de DSL Interna

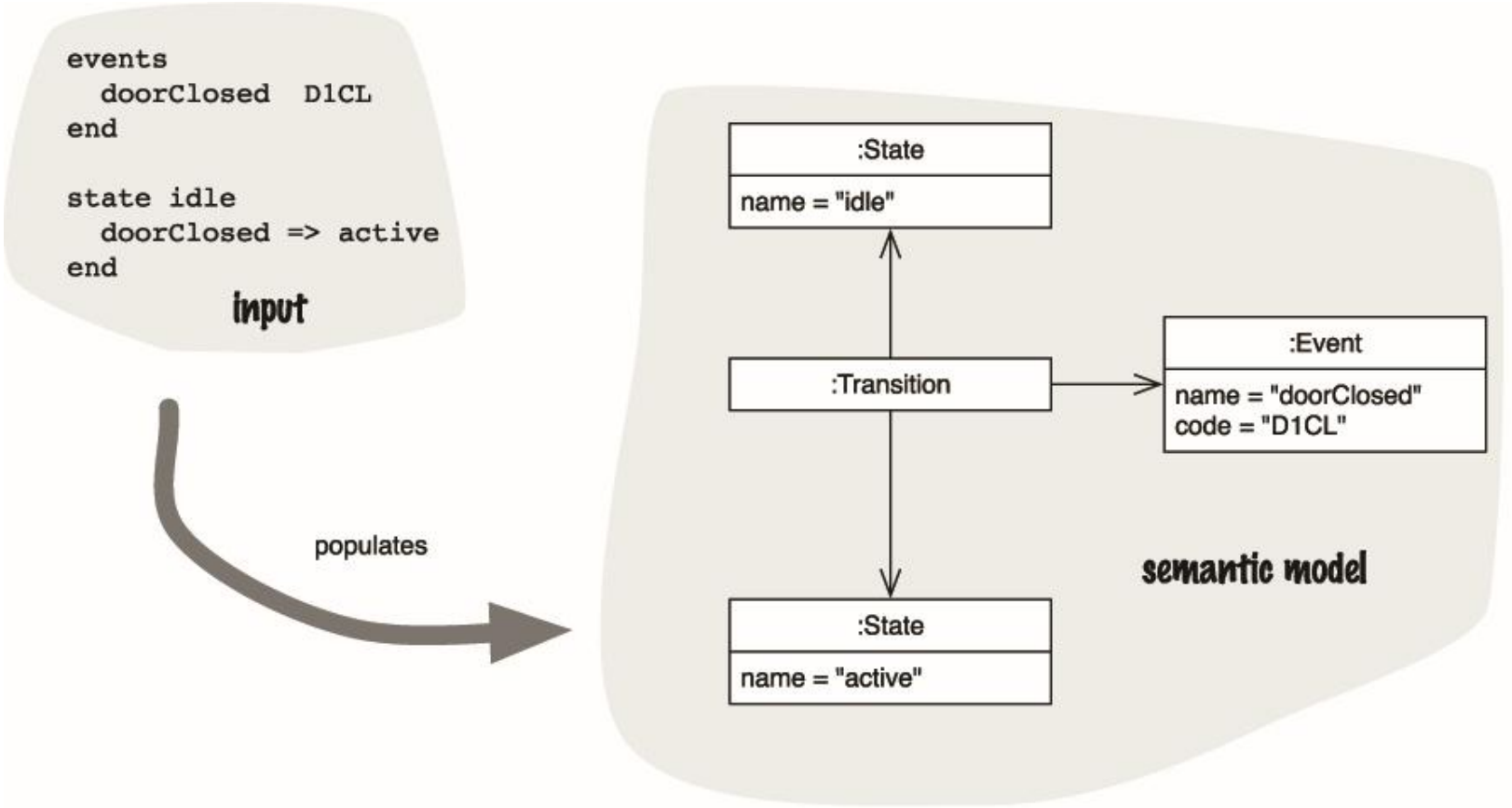
```
public class Customer {
    ...
    public OrderBuilder newOrder() {
      return new OrderBuilder(this);
    }
}
```

Exemplo de DSL Interna

```
public class OrderBuilder {
  // ...

  public OrderBuilder(Customer customer) {
    this.customer = customer;
    this.order = new Order();
  }

  public OrderBuilder with(int id, String name) {
     order.addProduct(new Product(id, name));
     return this;
  }

  public OrderBuilder priorityRush() {
    order.setPriorityRush(true);
    return this;
  }

  public void done() {
    customer.addOrder(this.order);
  }
```

Modelo Semântico

- DSLs devem ser usadas para popular um modelo semântico
- Modelo semântico: representação de entidades do domínio

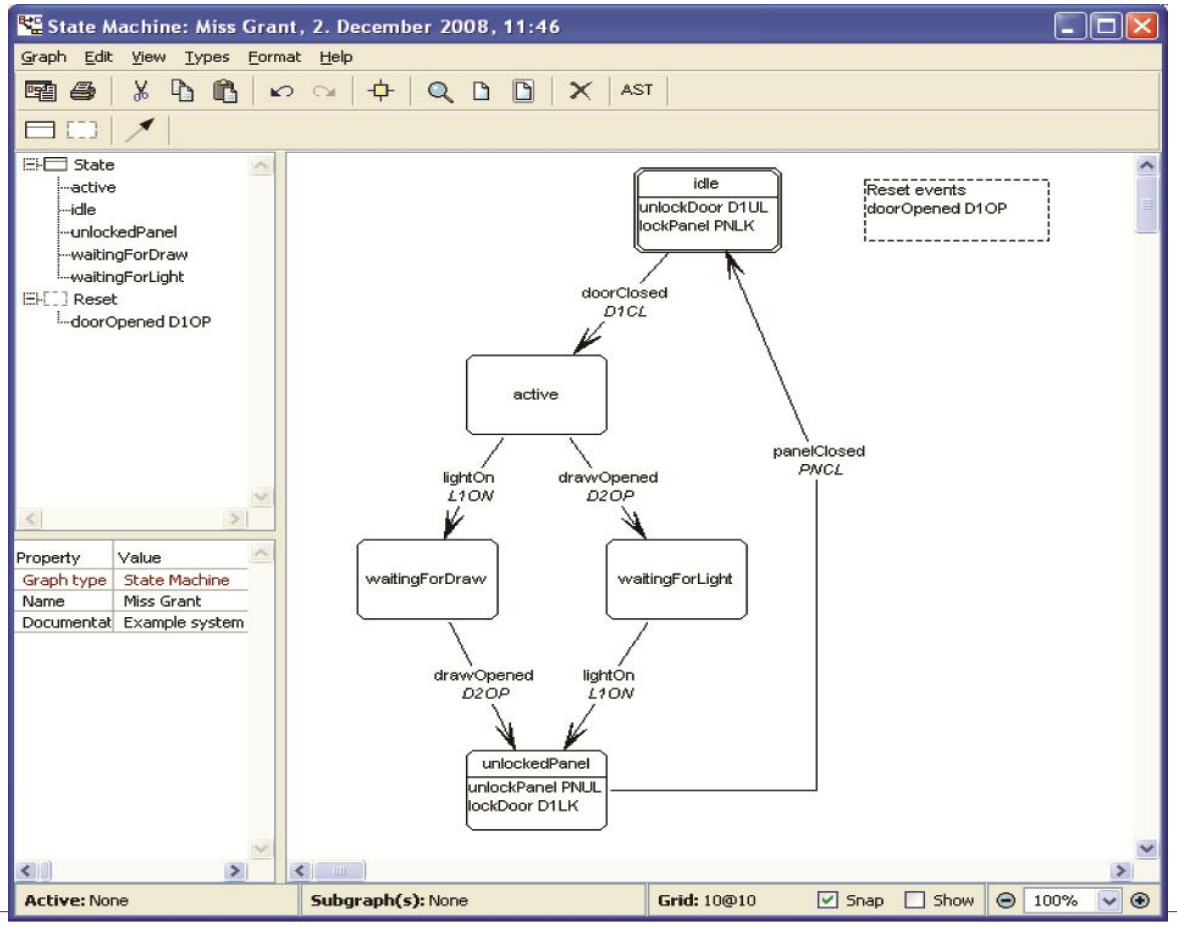


Modelo Semântico

- Modelo pode ser testado, analisado, estudado etc em separado
- Maior benefício: modelo bem planejado (e não a DSL em si)
- DSL é uma fachada para um modelo semântico do sistema alvo
- Processo de desenvolvimento:
  - Primeiro passo: Modelo
  - Segundo passo: DSL
- Comparação MDA/MDD:
  - Semelhança: modelo é a parte central do projeto
  - Diferença: não objetiva geração de código a partir do modelo

Language Workbenches

- Ambientes para facilitar a implementação de DSLs



Sugestões de Projetos

1. Sugestões anteriores envolvendo reimplementações de DCL

2. Implementação de uma DSL
   - Escolher domínio e modelo semântico
   - Usando XText

Arquitetura de Software para Aplicações Distribuídas

Middleware

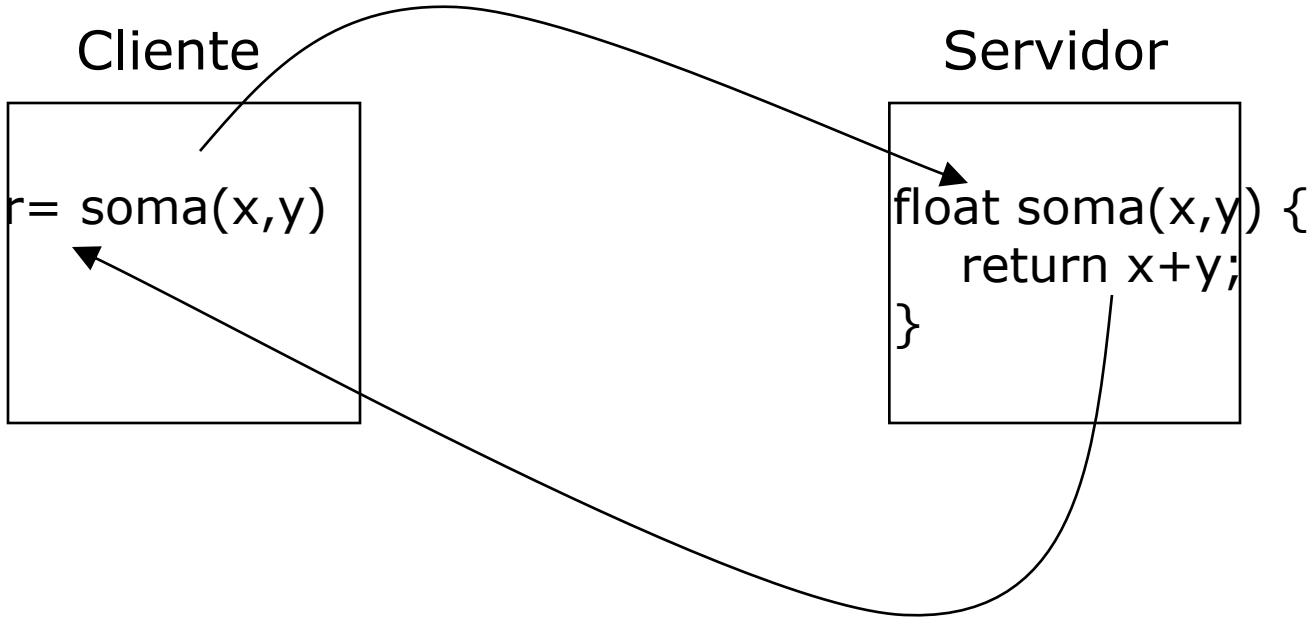
- Desenvolver uma aplicação distribuída é mais difícil do que uma aplicação centralizada
  - Problemas: comunicação, heterogeneidade, falhas, concorrência, segurança, escalabilidade etc
- Middleware: infra-estrutura de software que fica entre o sistema operacional e uma aplicação distribuída
- Objetivo: tornar mais simples o desenvolvimento de uma aplicação distribuída
- Ideia: oferecer abstrações de mais alto nível que tornem transparente detalhes de programação em redes
- Fazer com que programação distribuída seja semelhante a programação centralizada

Middleware

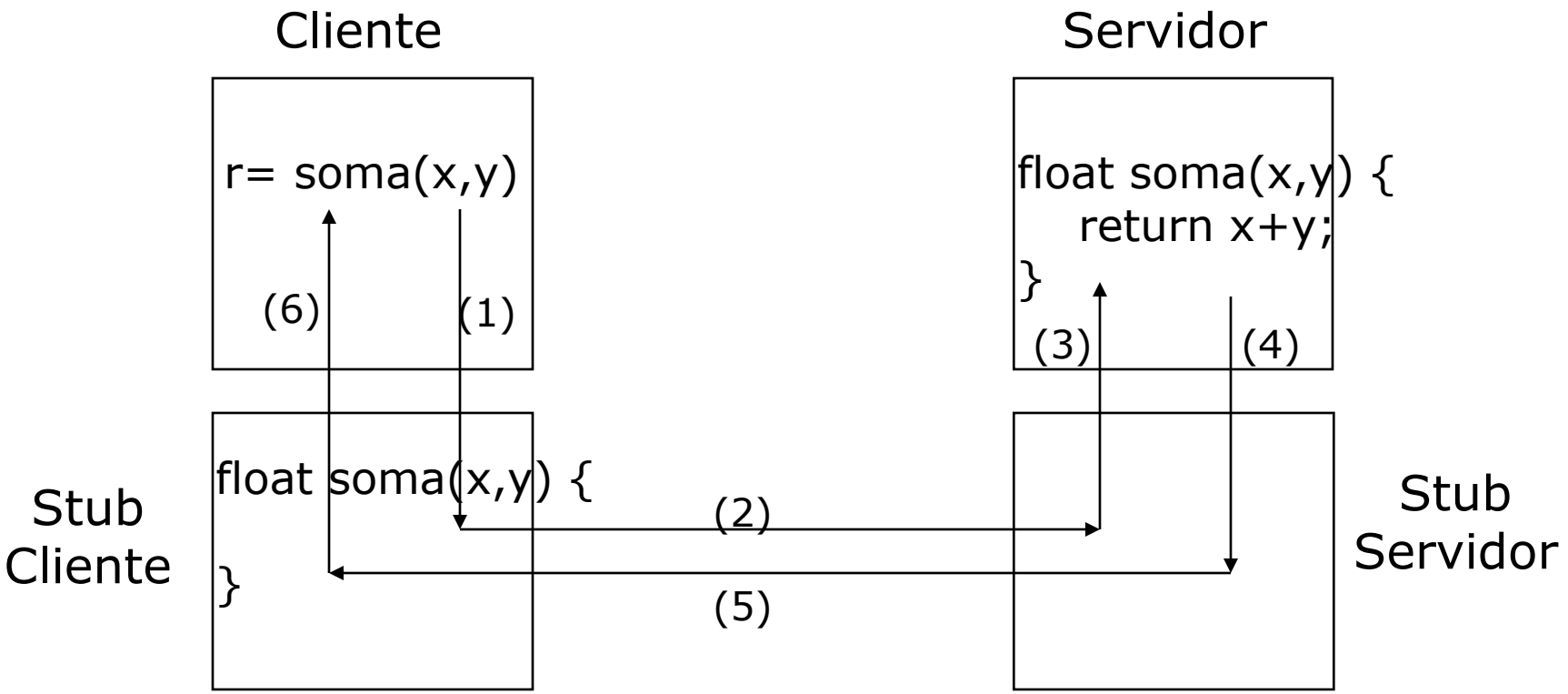
- Middleware e linguagens de programação:
  - LP: escondem o hardware (registradores, instruções de máquina, modos de endereçamento, periféricos etc)
  - Middleware: escondem a rede (endereços IP, portas, sockets, formato de mensagens, conexões, falhas etc)

- História sistemas de middleware:
  - RPC → linguagens estruturadas
  - RMI, CORBA, .NET Remoting → linguagens OO
  - Web Services → Web

RPC: Idéia Básica



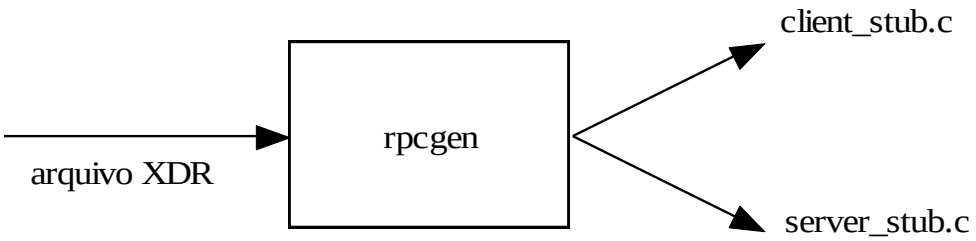
RPC: Arquitetura Interna



- Questão fundamental: Quem gera os stubs?
- Evidentemente, geração manual não faz sentido

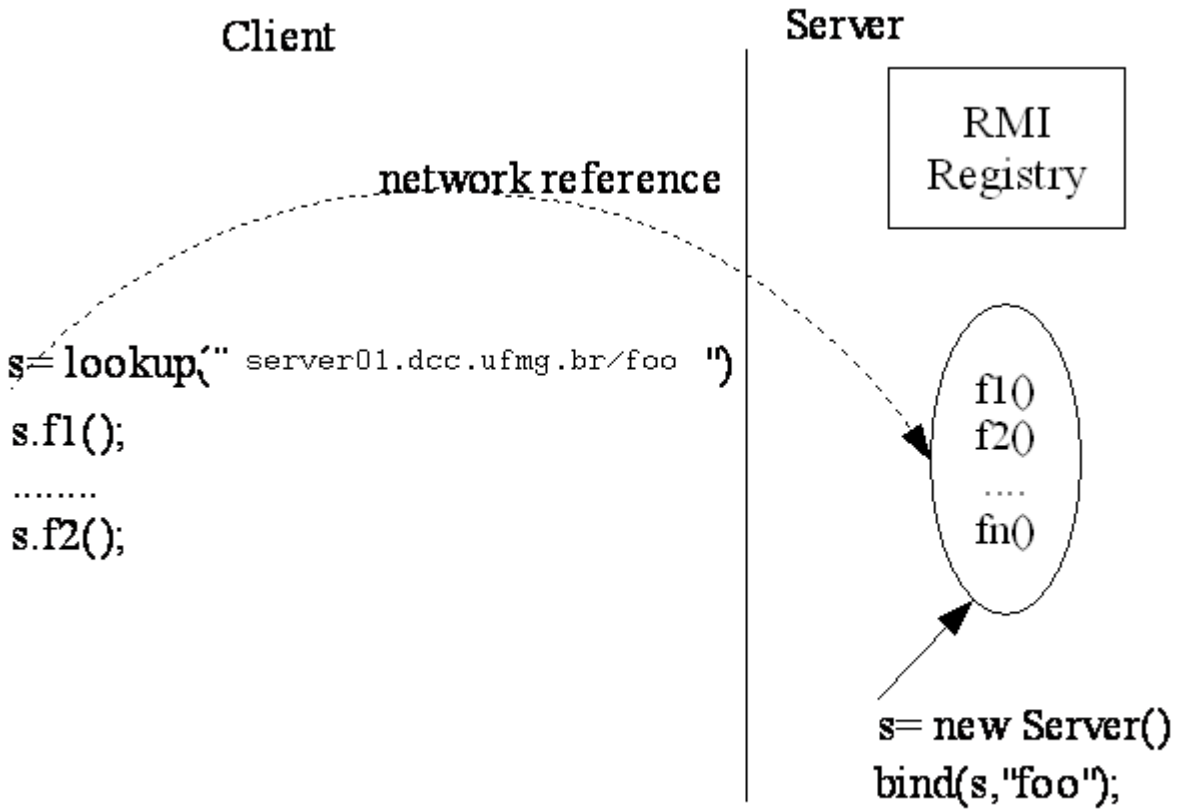
RPC: Arquitetura Interna

- Stubs são gerados automaticamente, por um compilador de stubs
- Entrada deste compilador: assinatura dos procedimentos remotos, em uma linguagem chamada XDR (External Data Representation)
  - Genericamente, conhecida como linguagem para definição de interfaces
- Saída deste compilador: código fonte dos stubs



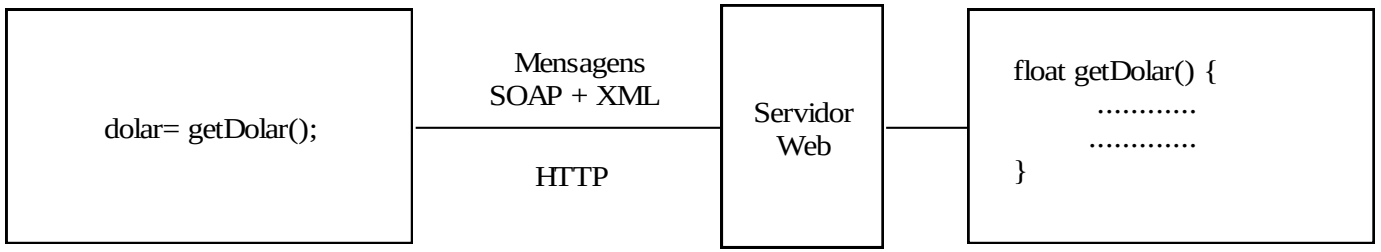
Java RMI

- Middleware para programação distribuída em Java
- Implementado como um pacote da linguagem



Serviços Web

- Extensão e adaptação do conceito de RMI para a *Web*
- Informações estruturadas, com interfaces bem definidas
- Principais padrões: HTTP, XML, SOAP, WSDL e UDDI
- Exemplo: serviço *Web* para obter cotação do dólar



SOA e SaaS

- Curso on-line de Berkley:
  - “Software Engineering for Software as a Service”
  - Professores: Armando Fox & David Patterson
  - https://www.coursera.org/saas

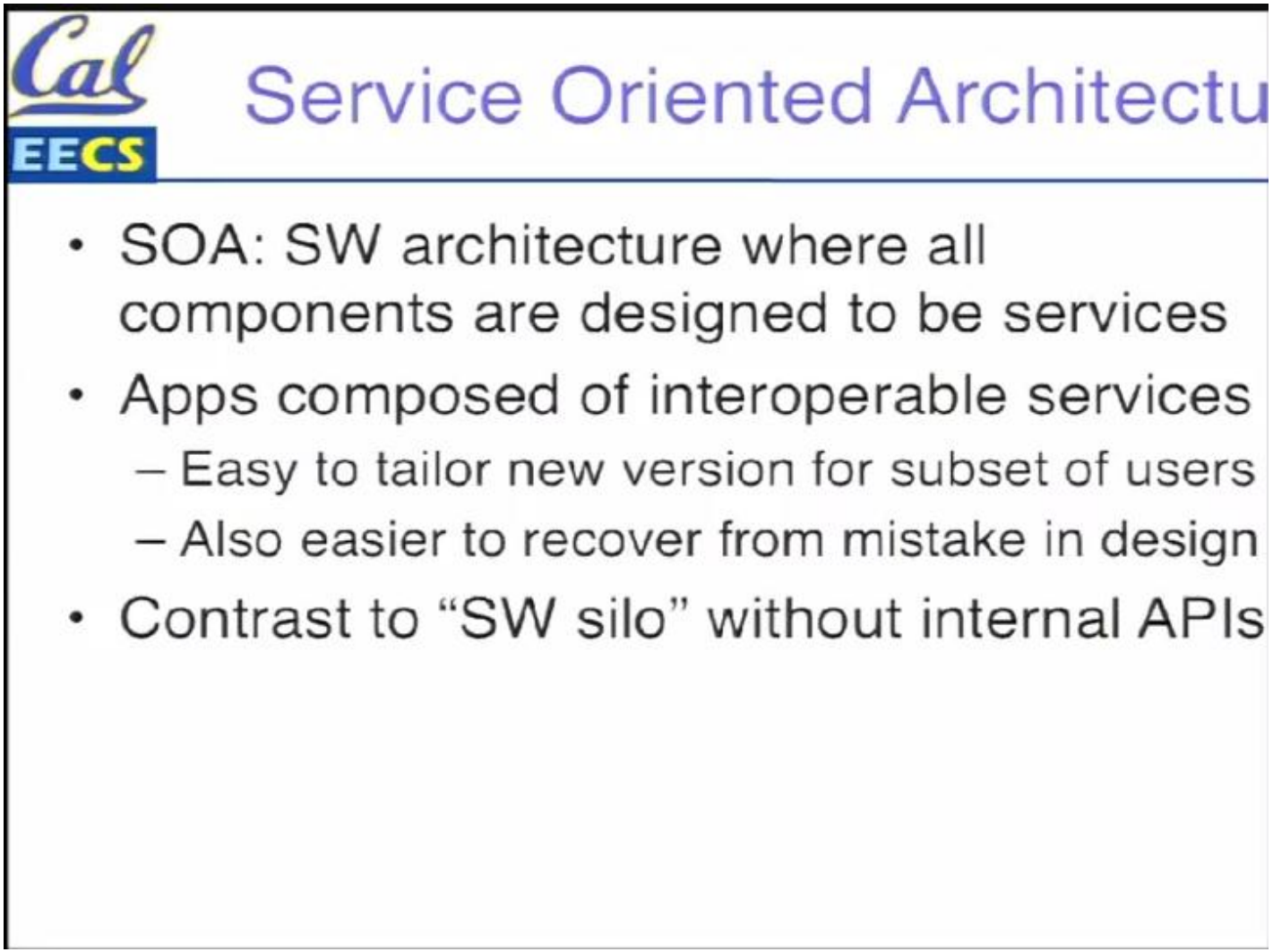
- Aulas:
  - 1.6 – Software as a Service
  - 1.7 – Service Oriented Architecture
  - 1.8, 1.9, 1.12 – Cloud Computing, Fallacies and Pitfalls

- Livro:
  - "Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing," Alpha Edition, by Armando Fox and David Patterson.

Software as a Service: SaaS

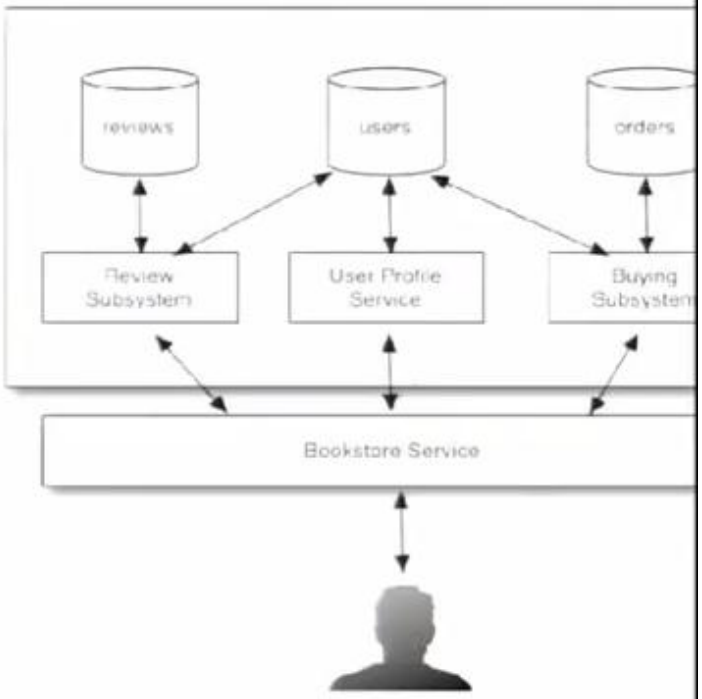
- Traditional SW: binary code installed and runs wholly on client device
- SaaS delivers SW & data as service over Internet via thin program (e.g., browser) running on client device
  - Search, social networking, video
- Now also SaaS version of traditional SW
  - E.g., Microsoft Office 365, TurboTax Online

58


Cal
EECS
Service Oriented Architectu
• SOA: SW architecture where all components are designed to be services
• Apps composed of interoperable services
– Easy to tailor new version for subset of users
– Also easier to recover from mistake in design
• Contrast to "SW silo" without internal APIs

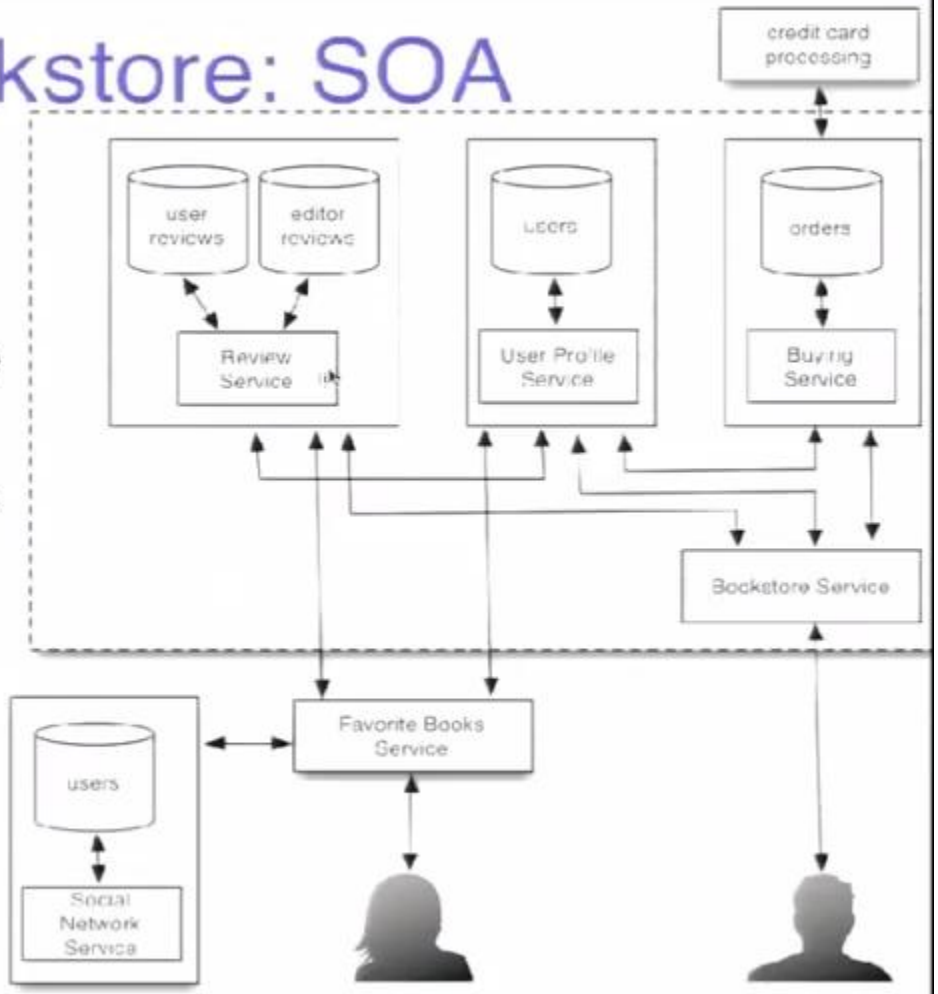
Bookstore: Silo

- Internal subsystems can share data directly
  - Review access user profile
- All subsystems inside single API ("Bookstore")



Bookstore: SOA

- Subsystems independent, as if in separate datacenters
  - Review Service access User Service API
- Can recombine to make new service ("Favorite Books")



Sugestões de Projeto

- Desenvolvimento de um “framework” baseado nos princípios de “Web Services / SOA / SaaS / Cloud Computing” para desenvolvimento ou integração de aplicações em um domínio particular (e de preferência bem novo)
  - Exemplos: smartphones, games, TV Digital, redes sociais etc

- Comparação entre “frameworks” (desenvolvimento de uma mesma aplicação em mais de um framework + comparação)

Exemplos de Papers

- Uma Arquitetura Orientada a Componentes para o Ginga, SBRC 2011
- Middleware Baseado em Componentes e Orientado a Recursos para Redes de Sensores sem Fio, SBRC 2012
- Model Driven RichUbi - A Model-Driven Process to Construct Rich Interfaces for Context Sensitive Ubiquitous Applications, SBES 2010
- Para mais ideias:
  - pesquisar papers no SBRC, SBES, SBCARS, WCGA