Programação Orientada a Objetos

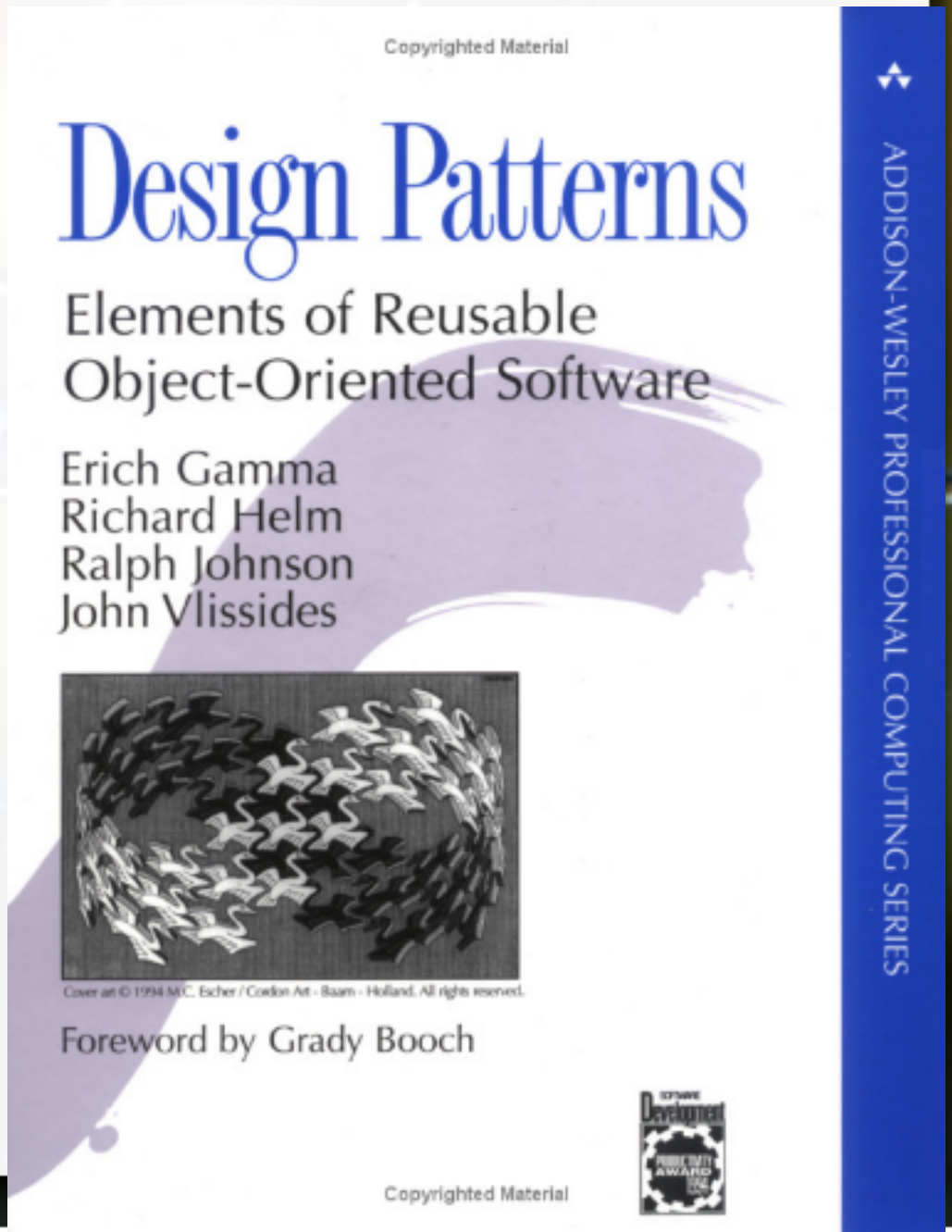
Padrões de Projeto

André Santanchè
Instituto de Computação – UNICAMP
Abril 2015

Padrões de Projeto

Design Patterns

- **Design Patterns: Elements of Reusable Object-Oriented Software**
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  - Addison-Wesley, 1995.

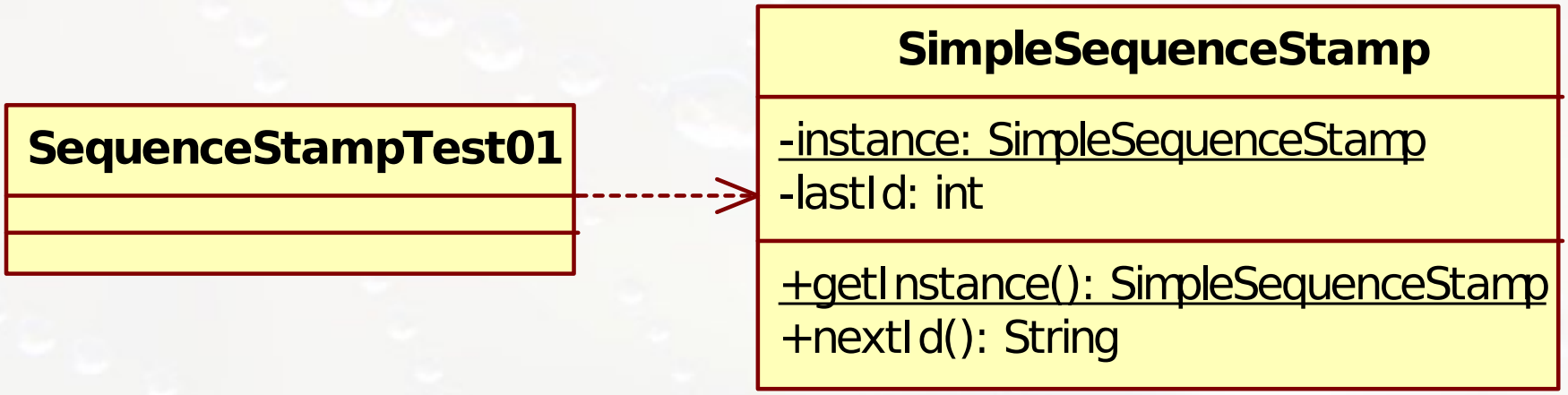


Padrões

- “cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira” (Alexander, 1977)

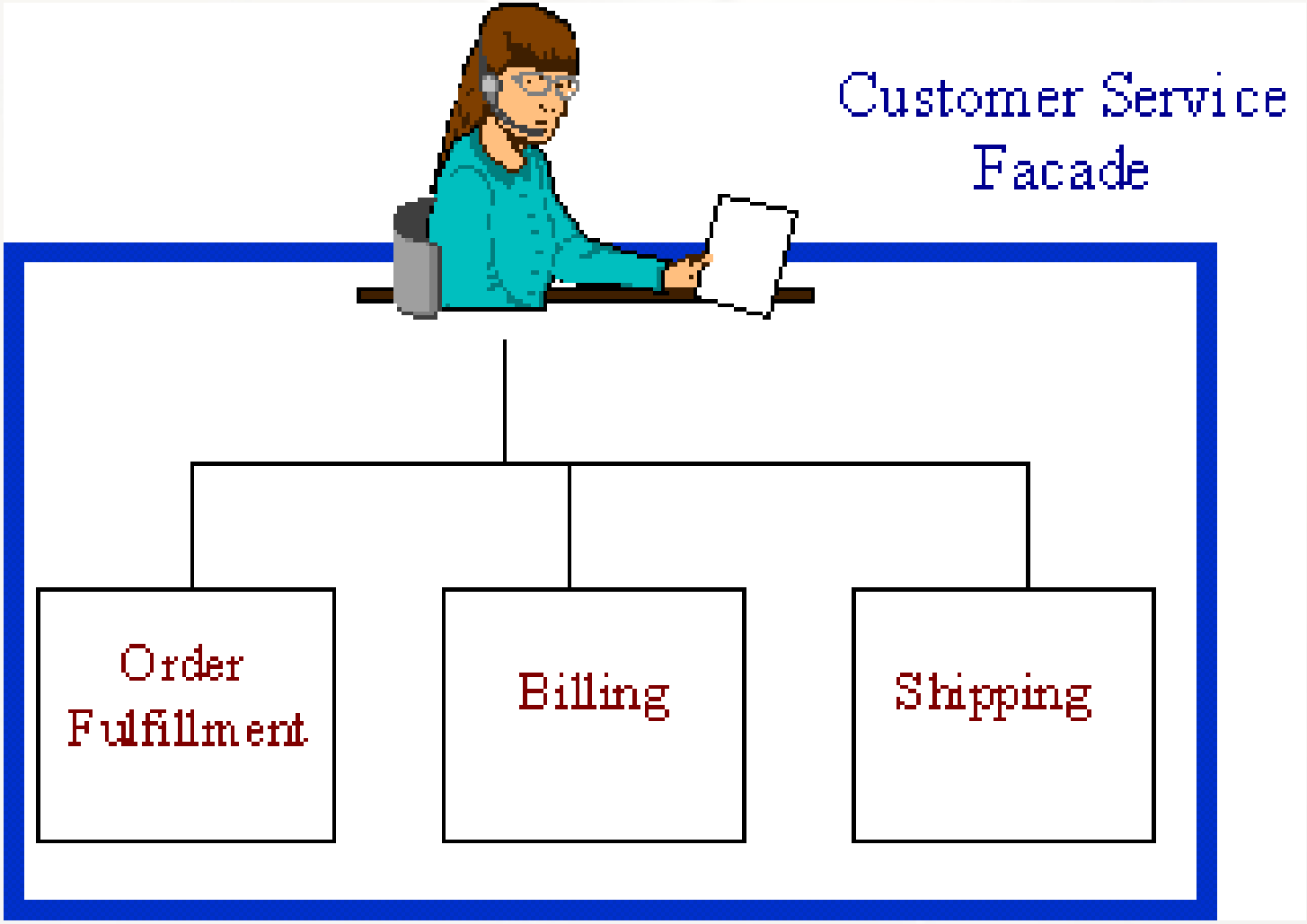
Pattern Singleton

Gerador de Identificador Seqüencial
Pattern Singleton



Pattern Facade

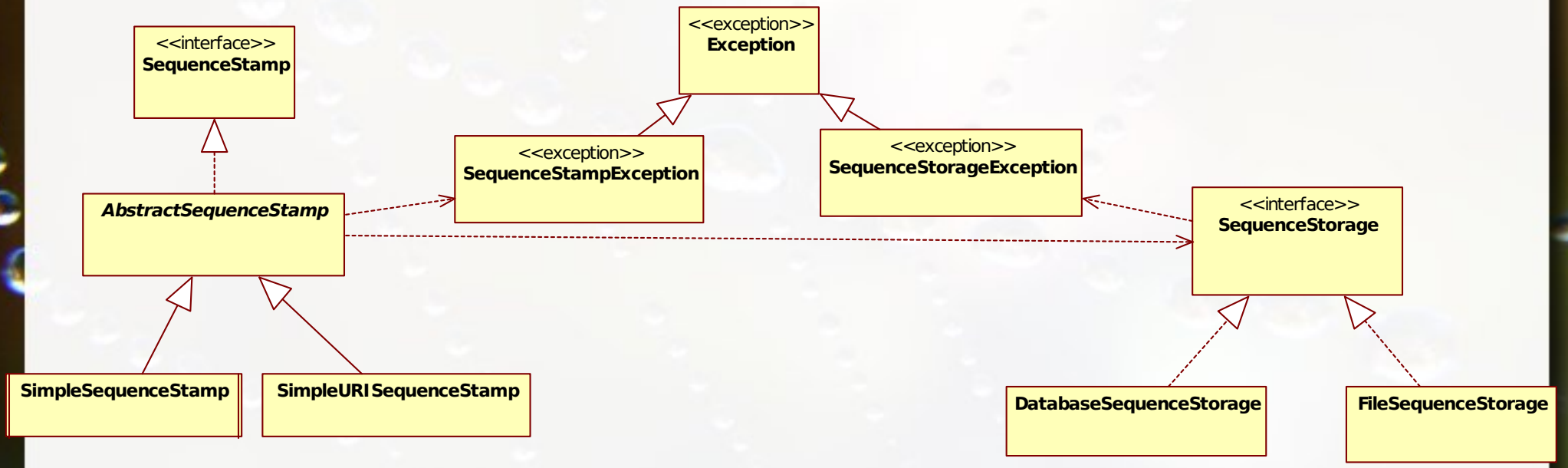
Facade



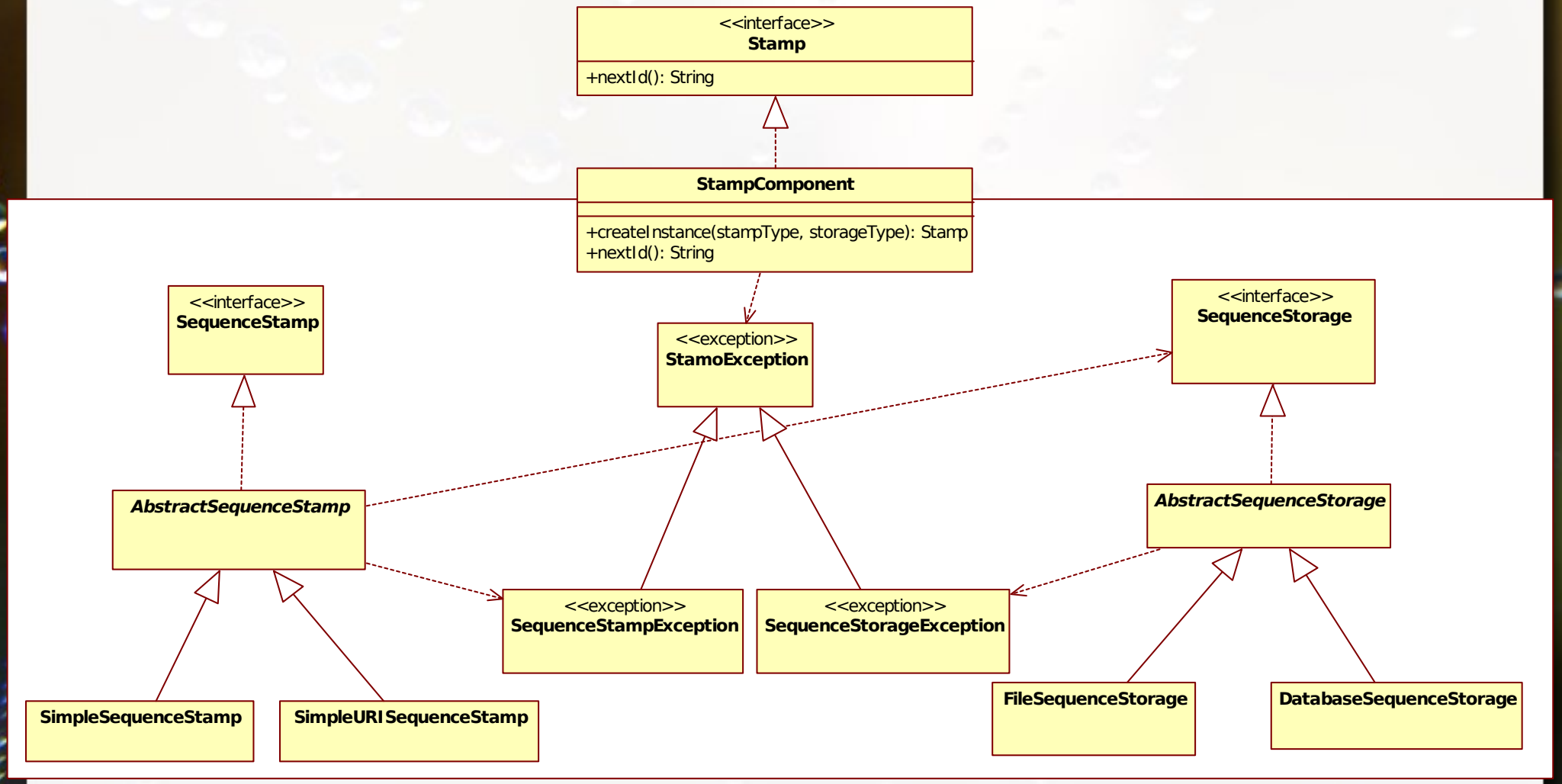
(AG Communication Systems, 1999)

Facade Pattern

Mini *Framework*



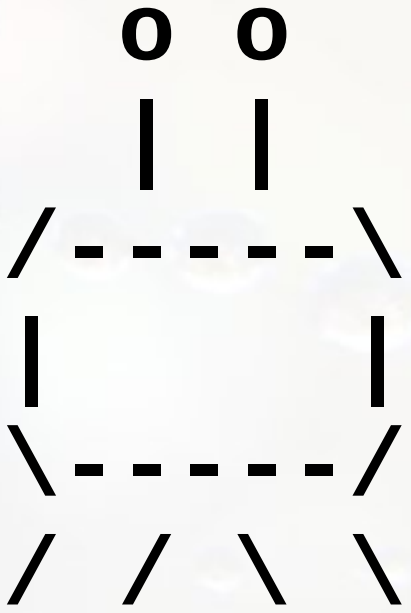
Facade Pattern
Interface Única



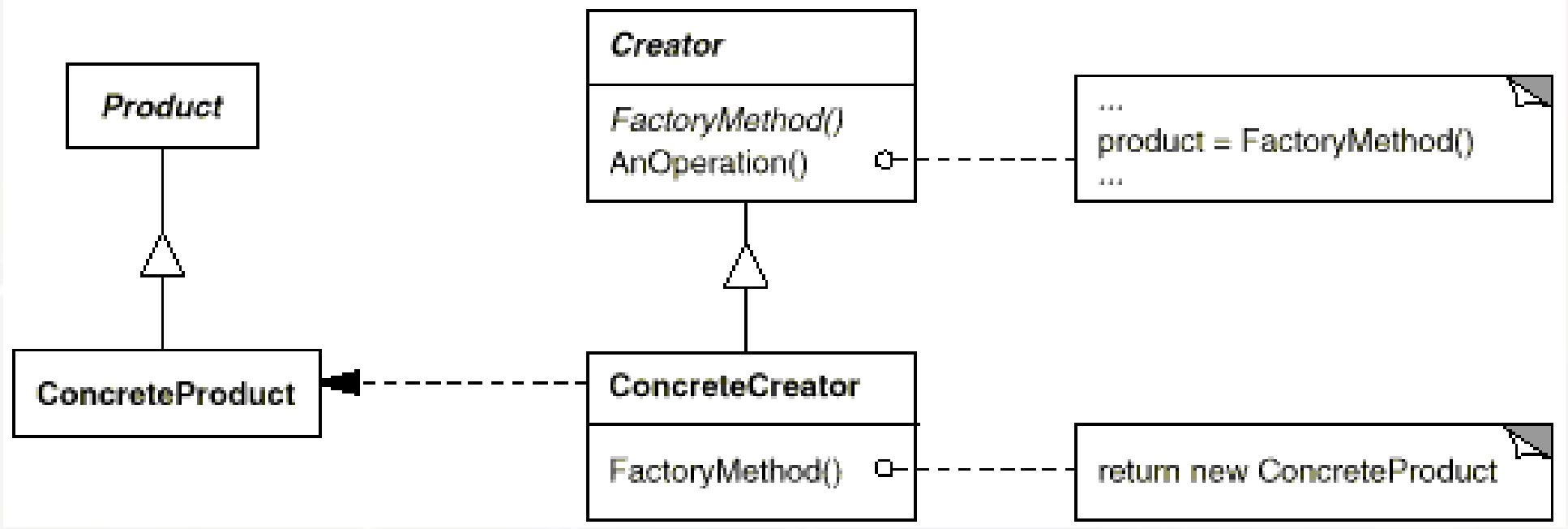
Pattern Factory

Fish DCC

- Goal
  - Draw a character-based Fish and Crab

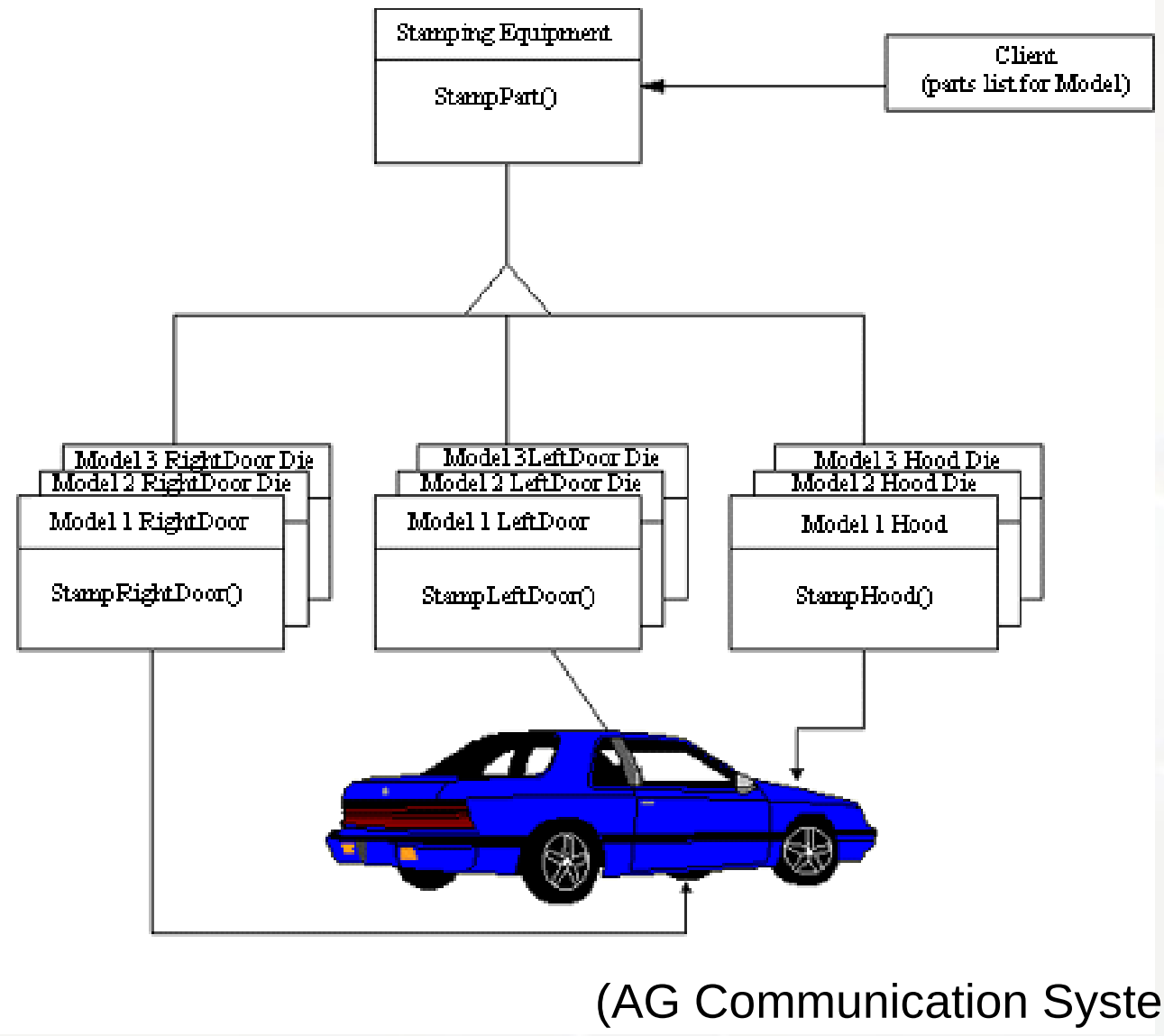


Padrões de Projeto
Factory Method



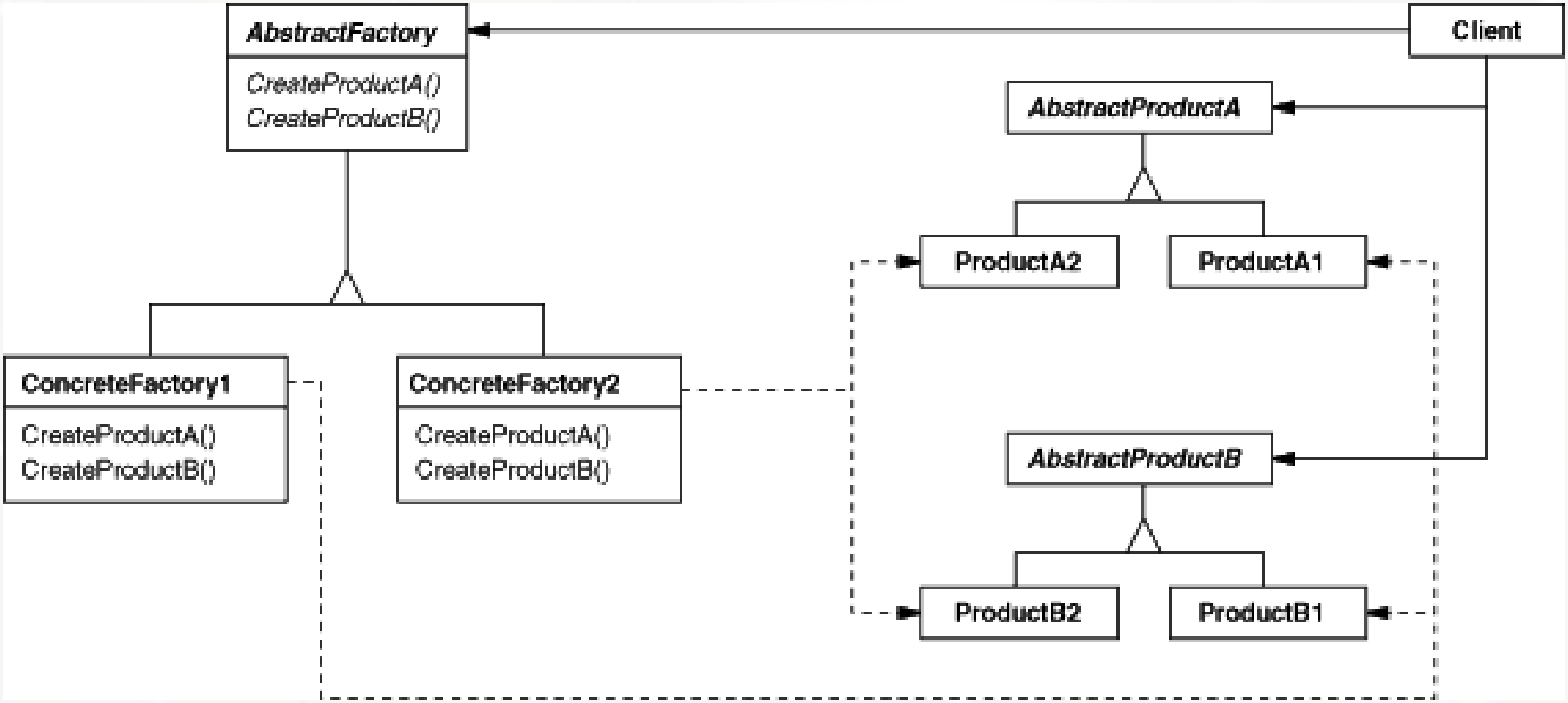
(Gamma, 1995)

Abstract Factory



(AG Communication Systems, 1999)

Padrões de Projeto
Abstract Factory



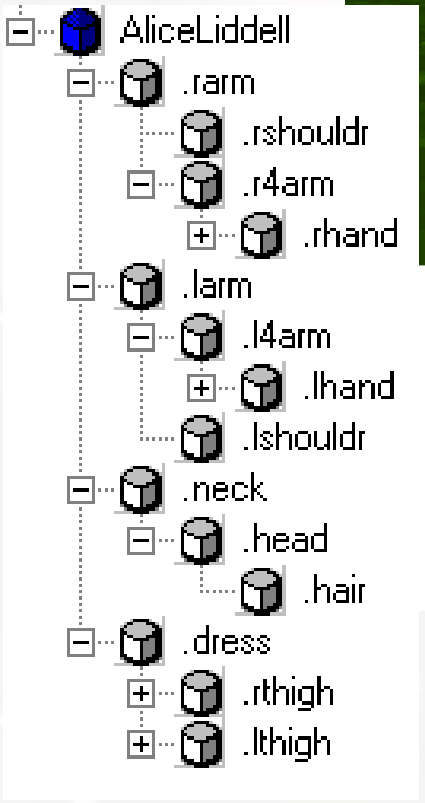
(Gamma, 1995)

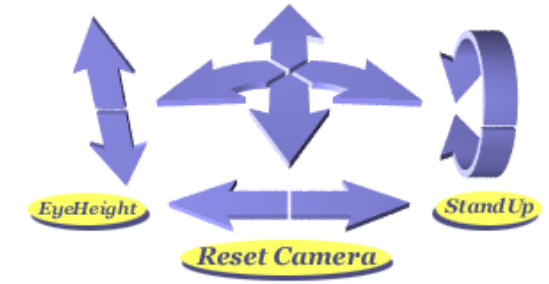
Pattern Composite

Alice

http://www.alice.org

- Ambiente 3D para a construção de animações/aplicações com propósitos educacionais
- Explora hierarquia de objetos





Alice

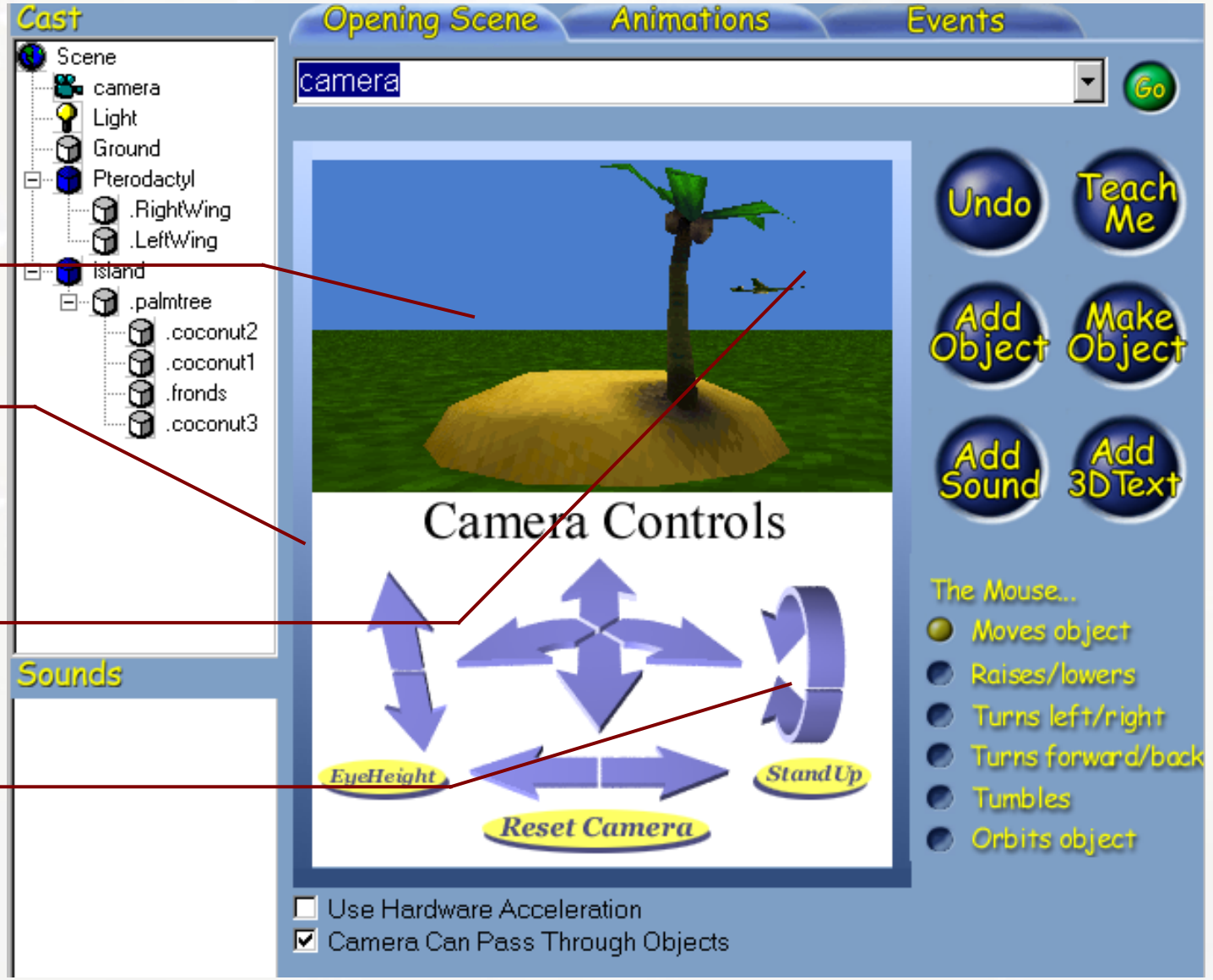
Objetos da Cena

Cena

Navegação no Espaço

Operações

Transform ações com o Objeto



Alice

Scripts associados a Objetos (comporta-mento)

Codificação “arrastando e soltando”

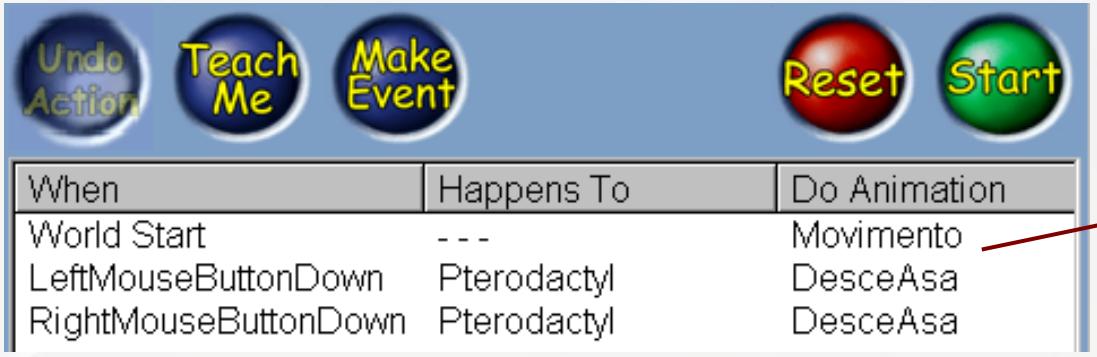
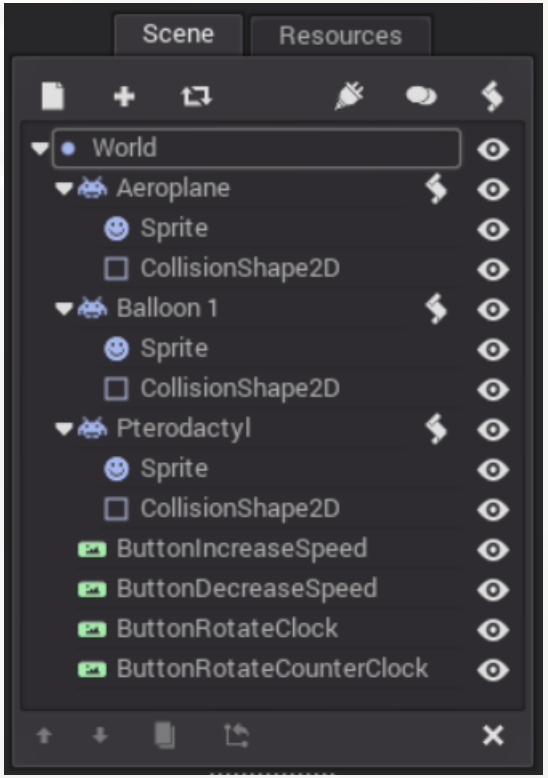


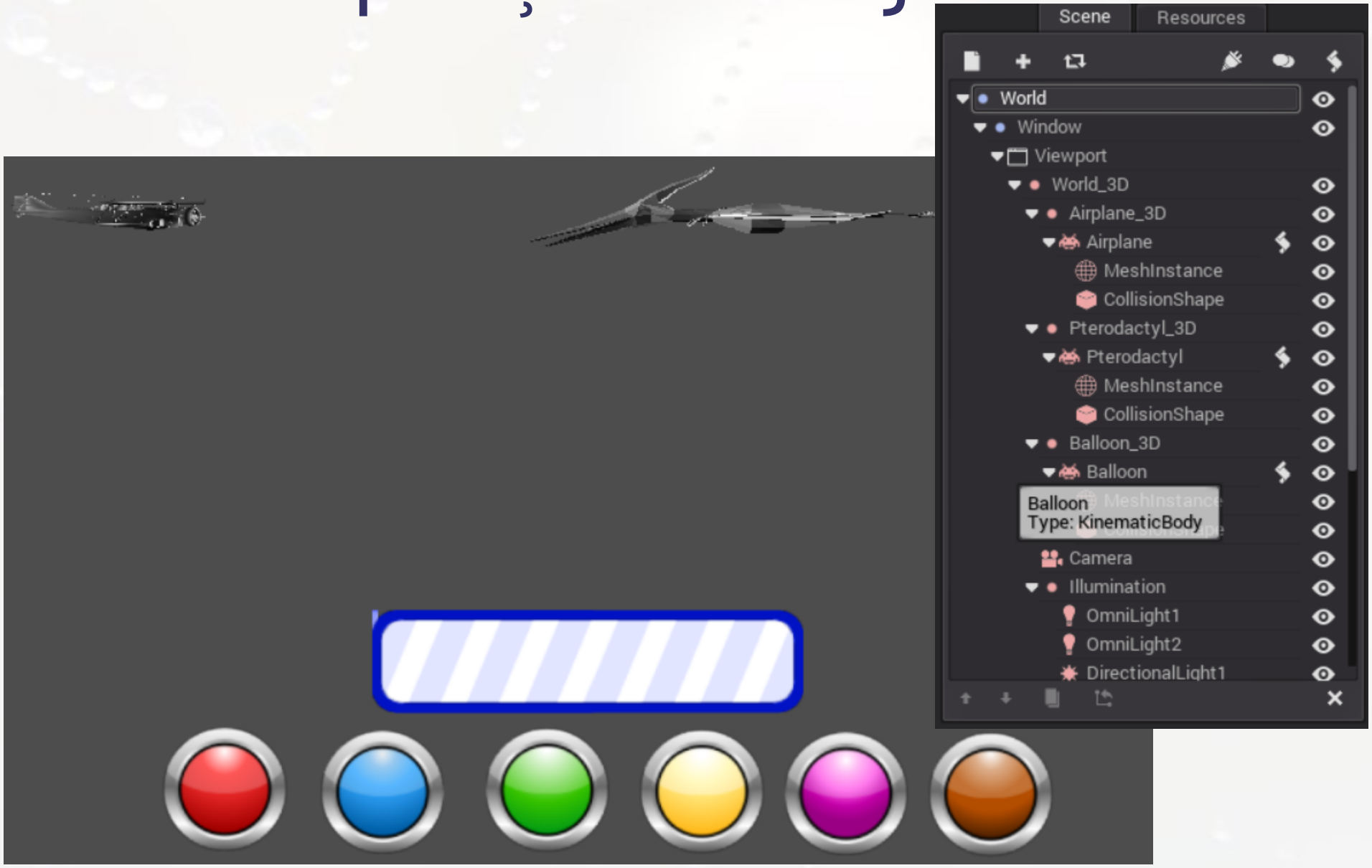
Tabela de Eventos

http://www.godotengine.org

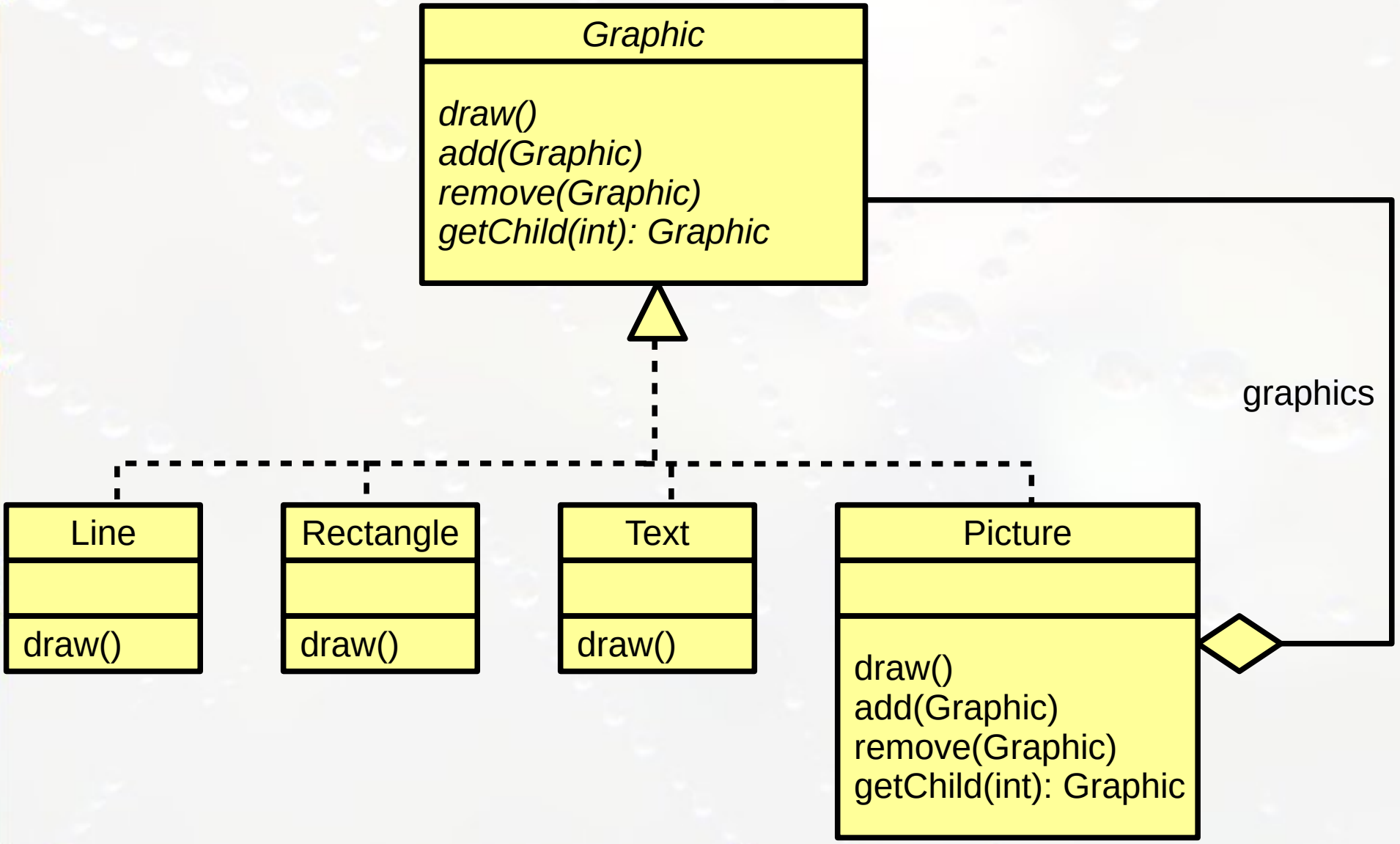
Godot
Composição de Objetos 2D



Godot
Composição de Objetos 3D



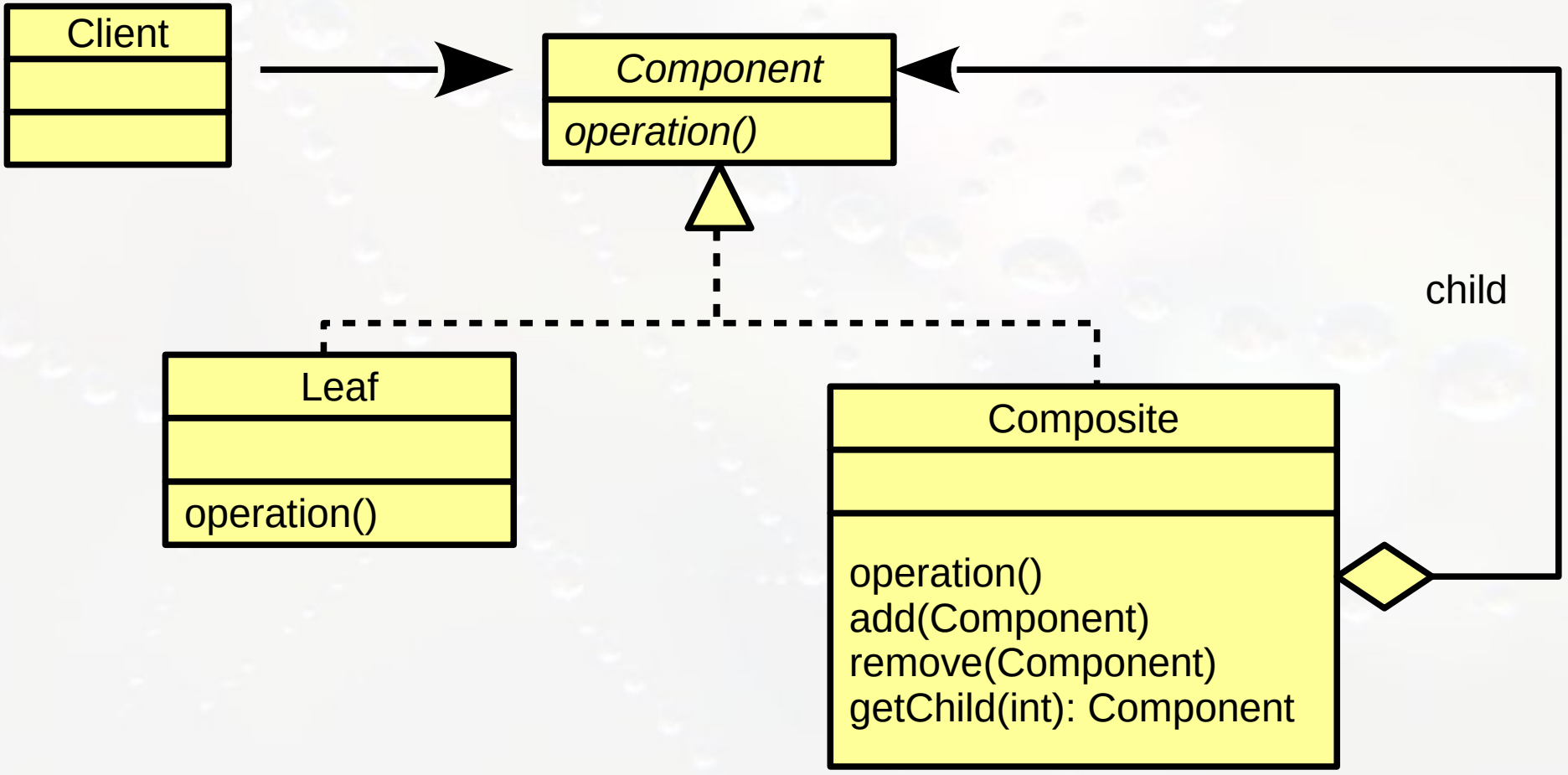
Hierarquia Gráfica



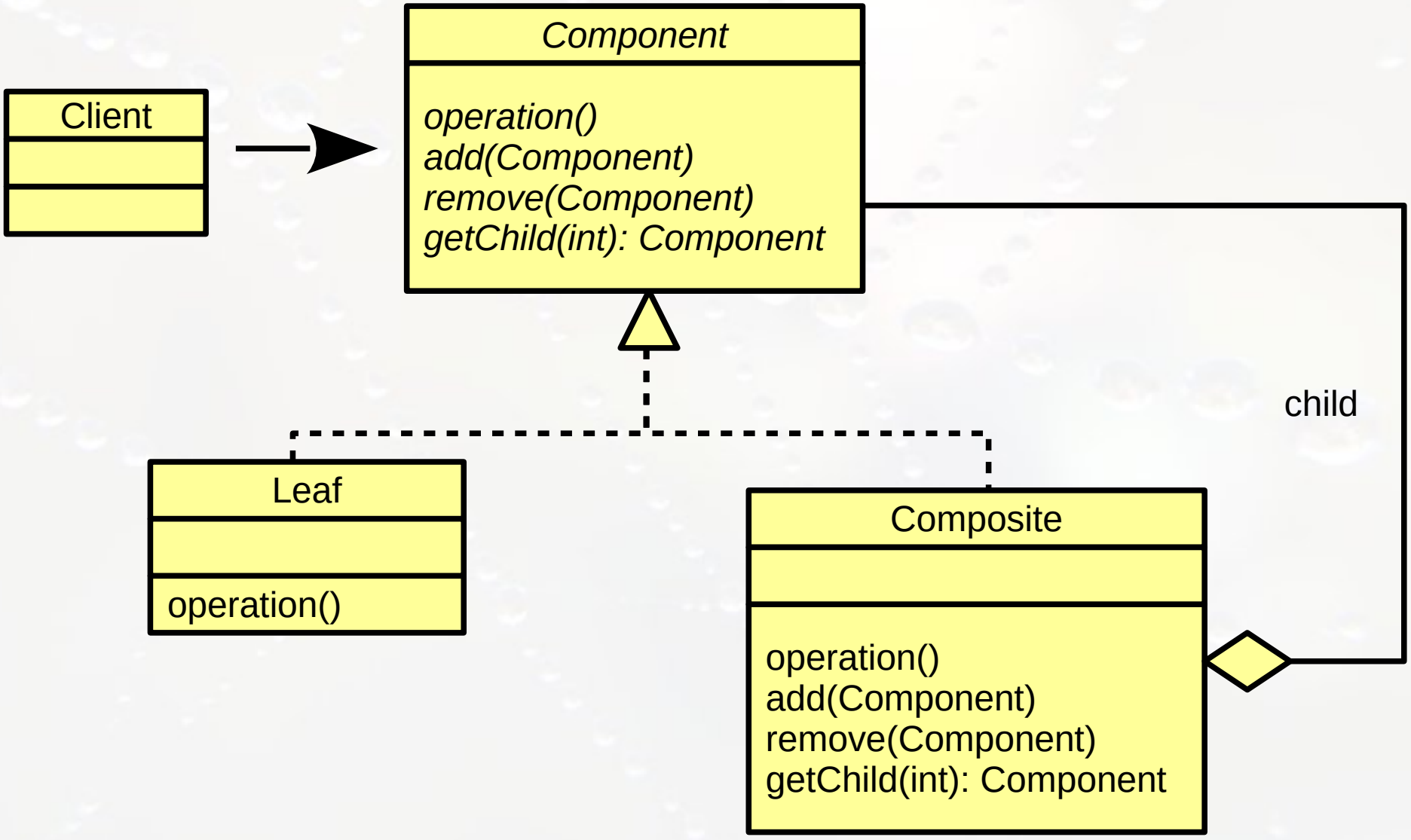
Composite

- Composição recursiva de objetos
- Ideal para representação parte/todo

Composite 1
Interface Mínima



Composite 2
Interface Máxima - Transparência



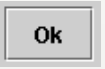
Exercício

- Quais as vantagens / desvantagens de usar Composite com interface Mínima ou Máxima?
- Sugestão: avalie aspectos de segurança e transparência

Composite
Segurança x Transparência

- Interface Mínima - Segurança
  - evita implementação de operações que não fazem sentido
  - por exemplo: **add**, **remove**, **getChild**
- Interface Máxima - Transparência
  - Cliente não precisa distinguir nós
  - **getChild** pode apenas retornar vazio
  - **add** e **remove** geram expectativa incorreta nas folhas

Swing - Componentes

Componente	Descrição	Exemplo	
Containers de alto nível	Componente principal que irá conter os demais	`JFrame`	
Containers intermediários	Pode conter outros componentes porém tem que estar inserido em um componente superior	`JPanel`	
Componentes atômicos	Componentes auto-suficientes que possuem uma apresentação e funcionalidade	`JButton`	

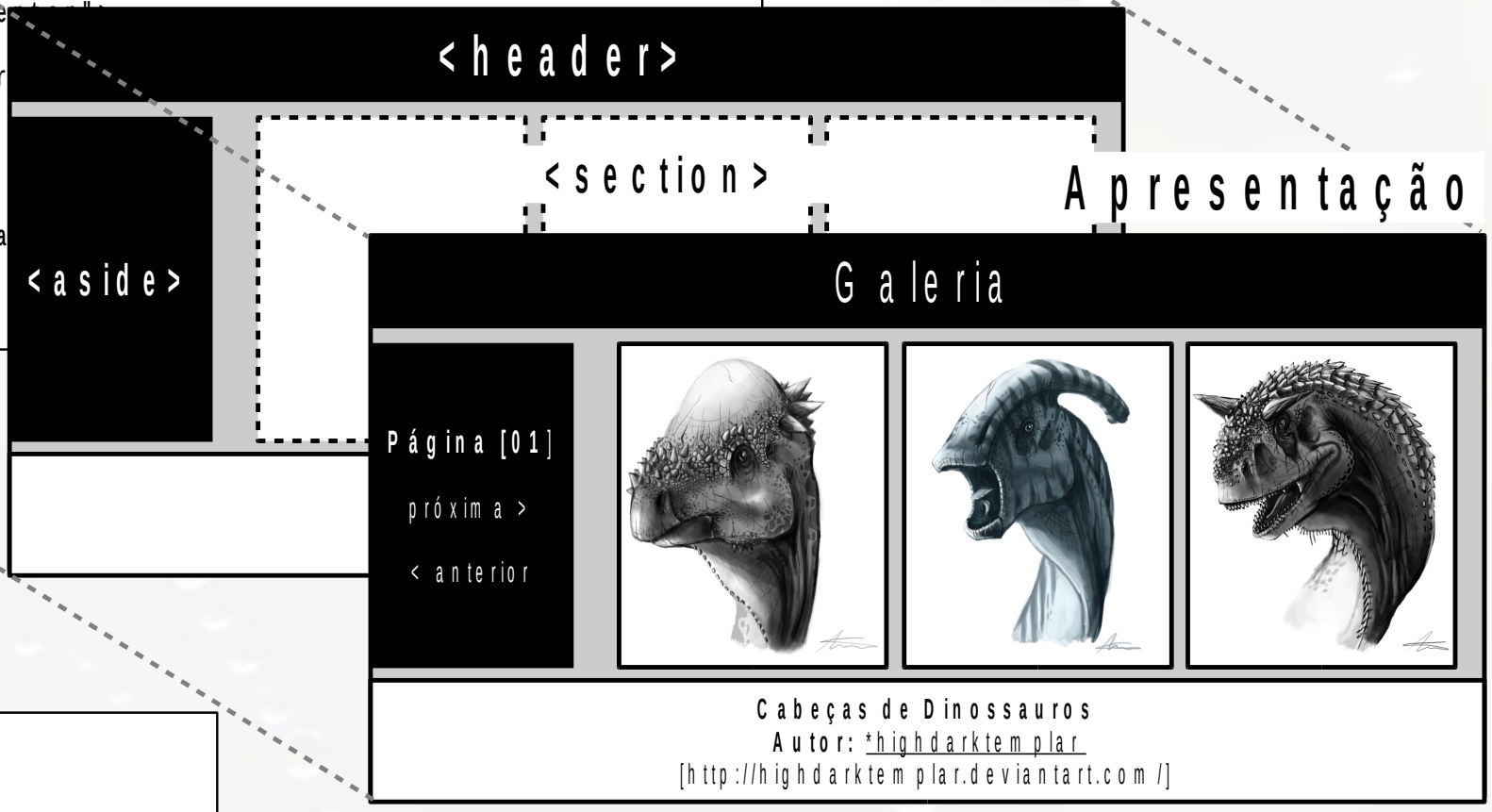
DOM - Document Object Model

- API de objetos para documentos XML e HTML
- Definido em CORBA IDL, ECMAScript e Java
- Organizado em níveis
- Nível 1:
  - DOM Core – funcionalidades básicas para documentos XML
  - DOM HTML – sobre o DOM Core → acrescenta funcionalidades para HTML

HTML

```
<header> Galeria </header>
<section>
    <aside>
        Página [01]
        …
    </aside>
    <section class="ce
        ...
        <figure><img sr
        ...
    </section>
</section>
<footer>
    Cabeças de Dinossa
     ...
</footer>
```

CSS

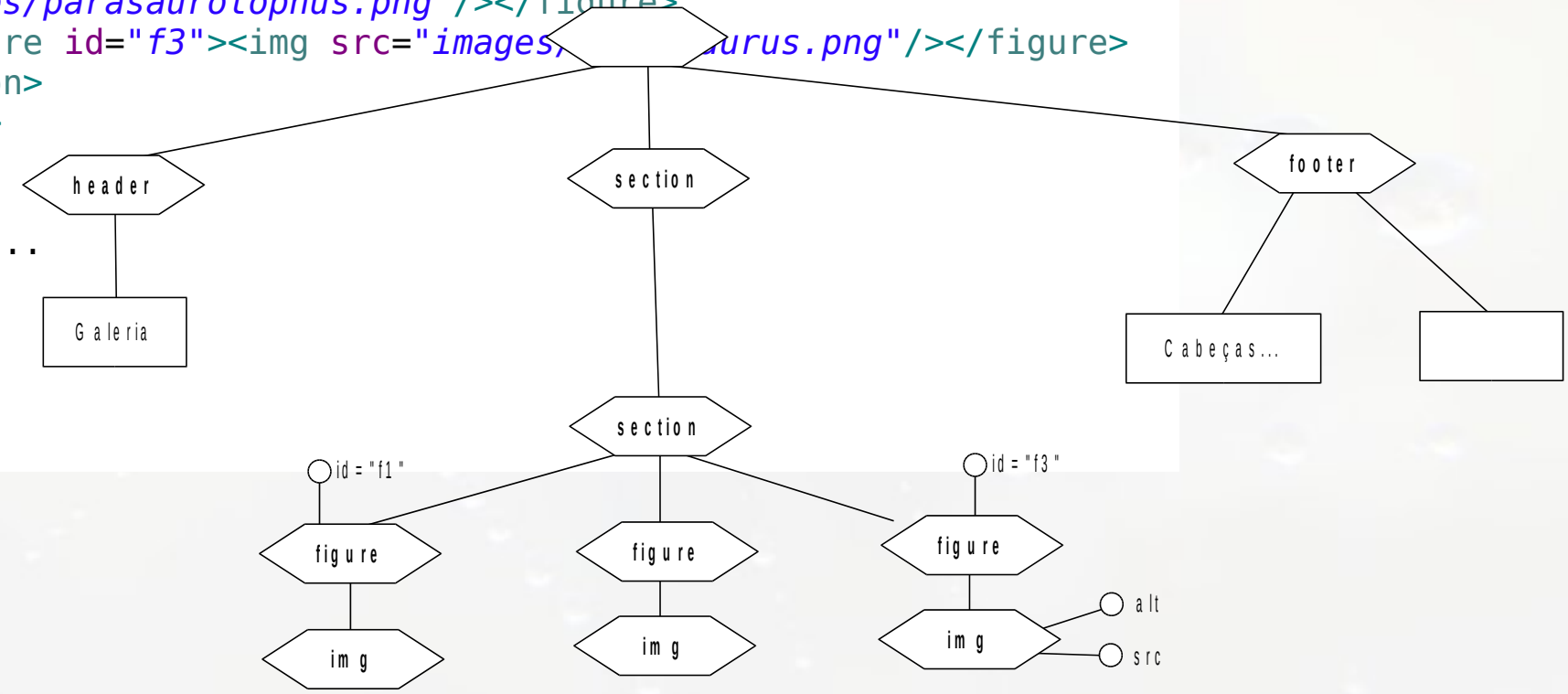


Apresentação

- **Cabeçalho**
  - Galeria
- **Secundário**
  - Número da página
  - Navegação: próximo/anterior
- **Central**
  - Sequência de imagens
- **Rodapé**
  - Dados do autor

Plano

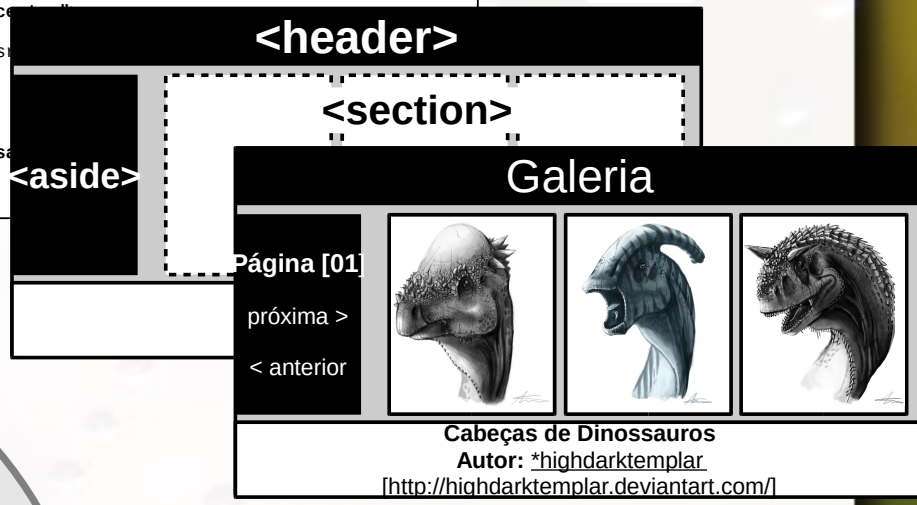
```
<html>
...
<body>

<header>
  Galeria
</header>

<section class="main">
  …
  <section class="center">
     <figure id="f1"><img
src="images/pachycephalosaurus.png"/></figure>
     <figure id="f2"><img
src="images/parasaurolophus.png"/></figure>
     <figure id="f3"><img src="images/[illegible]urus.png"/></figure>
  </section>
</section>

<footer>
  Cabeças...
</footer>

</body>

</html>
```



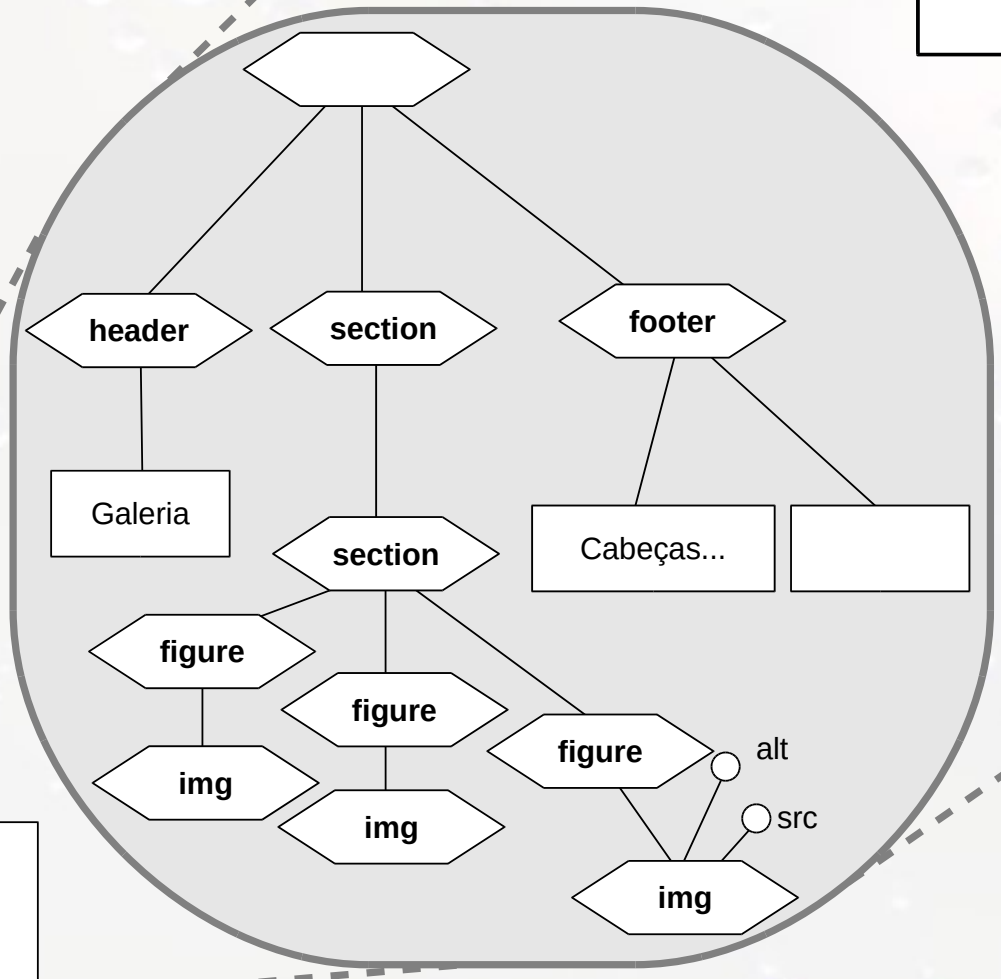
HTML

```
<header> Galeria </header>
<section>
   <aside>
      Página [01]
      …
   </aside>
   <section class="
      ...
      <figure><img s
      ...
   </section>
</section>
<footer>
   Cabeças de Dinossa
    ...
</footer>
```

CSS



DOM

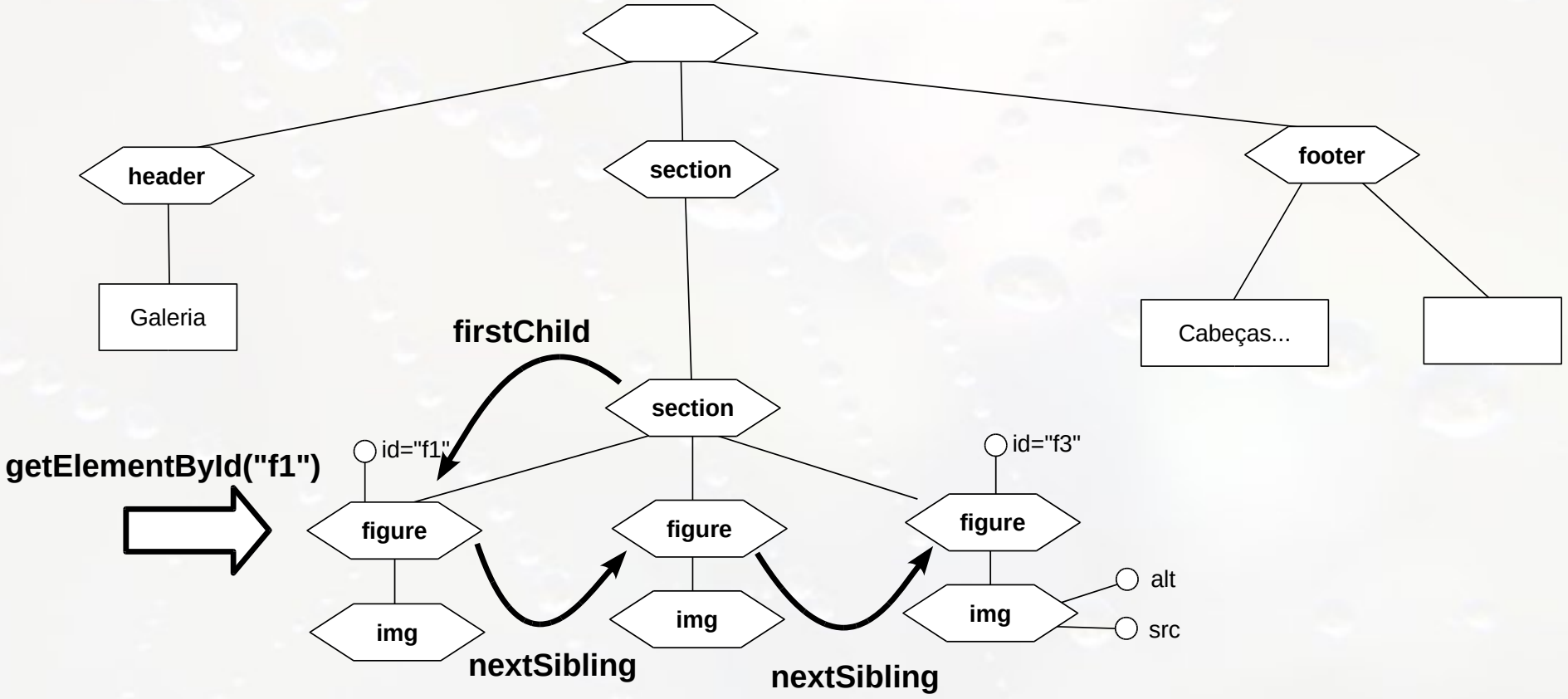


Aplicativo JavaScript

Navegando pelo DOM

- **Node** – genericamente qualquer nó
- **Element** – elementos HTML/XML representados por tags
- **Attr** – atributos associados a elementos.
- **Text** – conteúdo texto livre
- **Document** – Nó raiz da árvore que representa o documento completo

Navegando pelo DOM



SVG – Scalable Vector Graphics

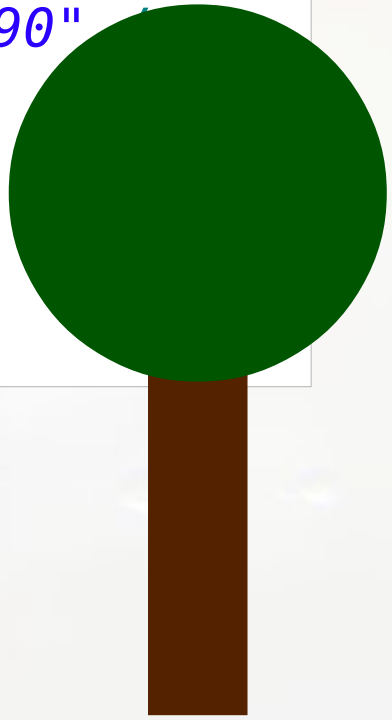
- Formato XML para a representação de imagens vetoriais (Dahlström et al., 2011)
- Suporte nativo dos navegadores

Primitivas SVG

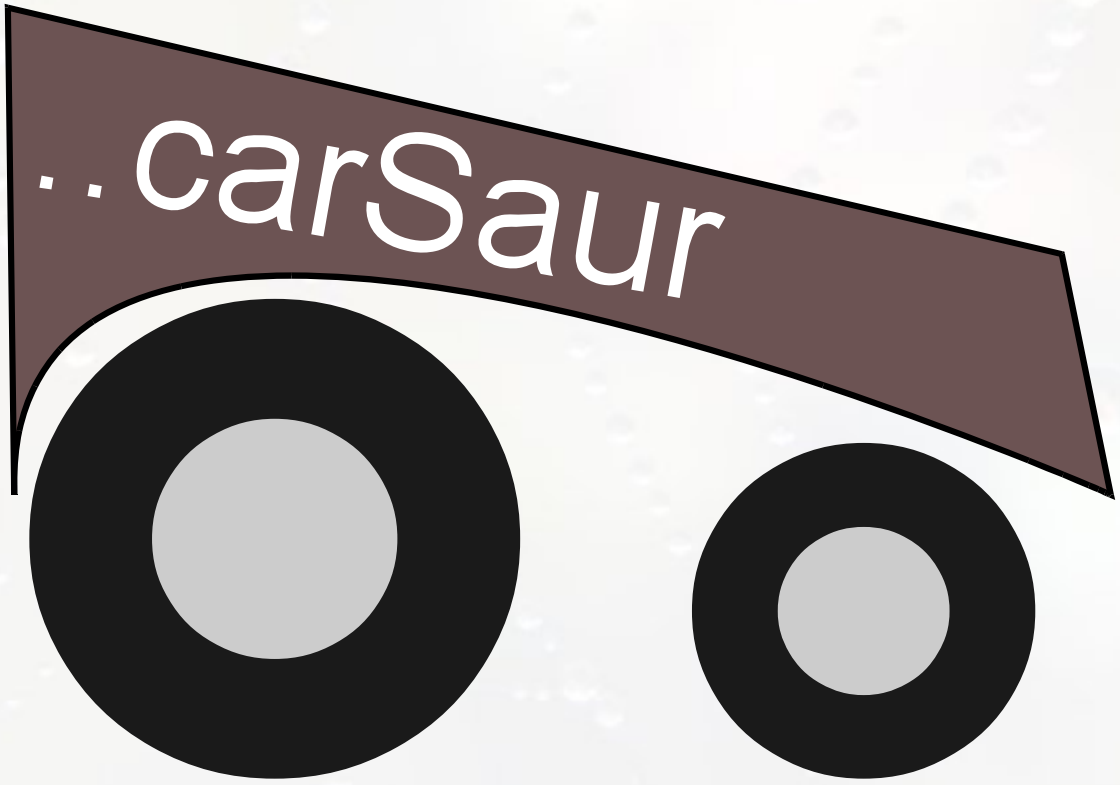
Primitiva	Descrição	Atributos	
<rect>	Desenha um retângulo.	style	Estilo de apresentação. Neste caso define a cor de preenchimento.
		x, y	Coordenadas do canto esquerdo superior.
		width, height	Altura e largura do retângulo.
<circle>	Desenha um círculo.	style	Estilo de apresentação. Neste caso define a cor de preenchimento.
		cx, cy	Coordenadas do centro do círculo.
		r	Raio do círculo.

Árvore SVG

```
<svg id="desenho" version="1.1"
     xmlns="http://www.w3.org/2000/svg"
     width="205" height="370">

    <rect style="fill:#552200"
          x="77" y="179" width="50" height="190" />

    <circle style="fill:#005500"
            cx="102" cy="106" r="95" />

</svg>
```



Carro SVG



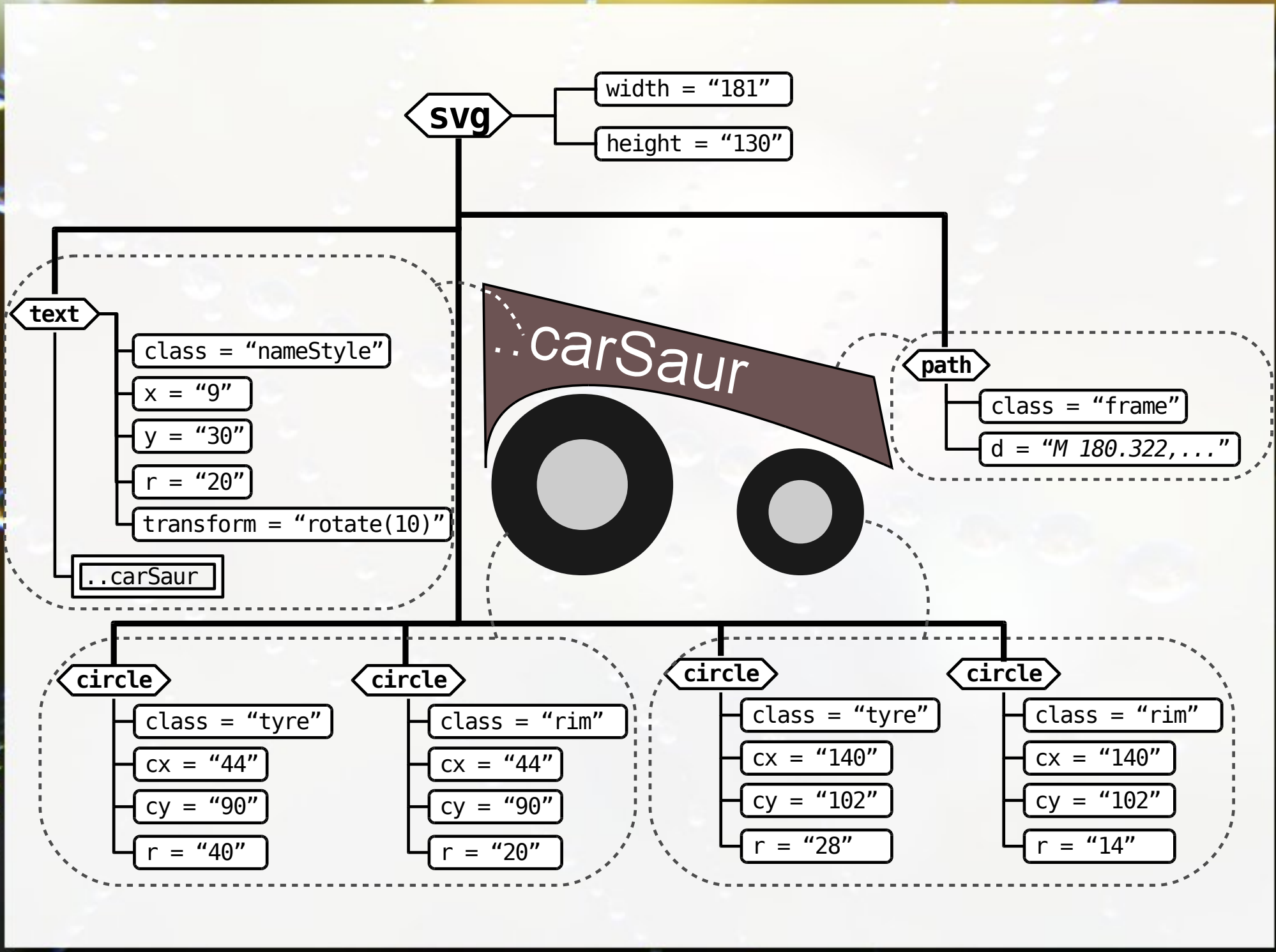
Mais Primitivas SVG para o Carro

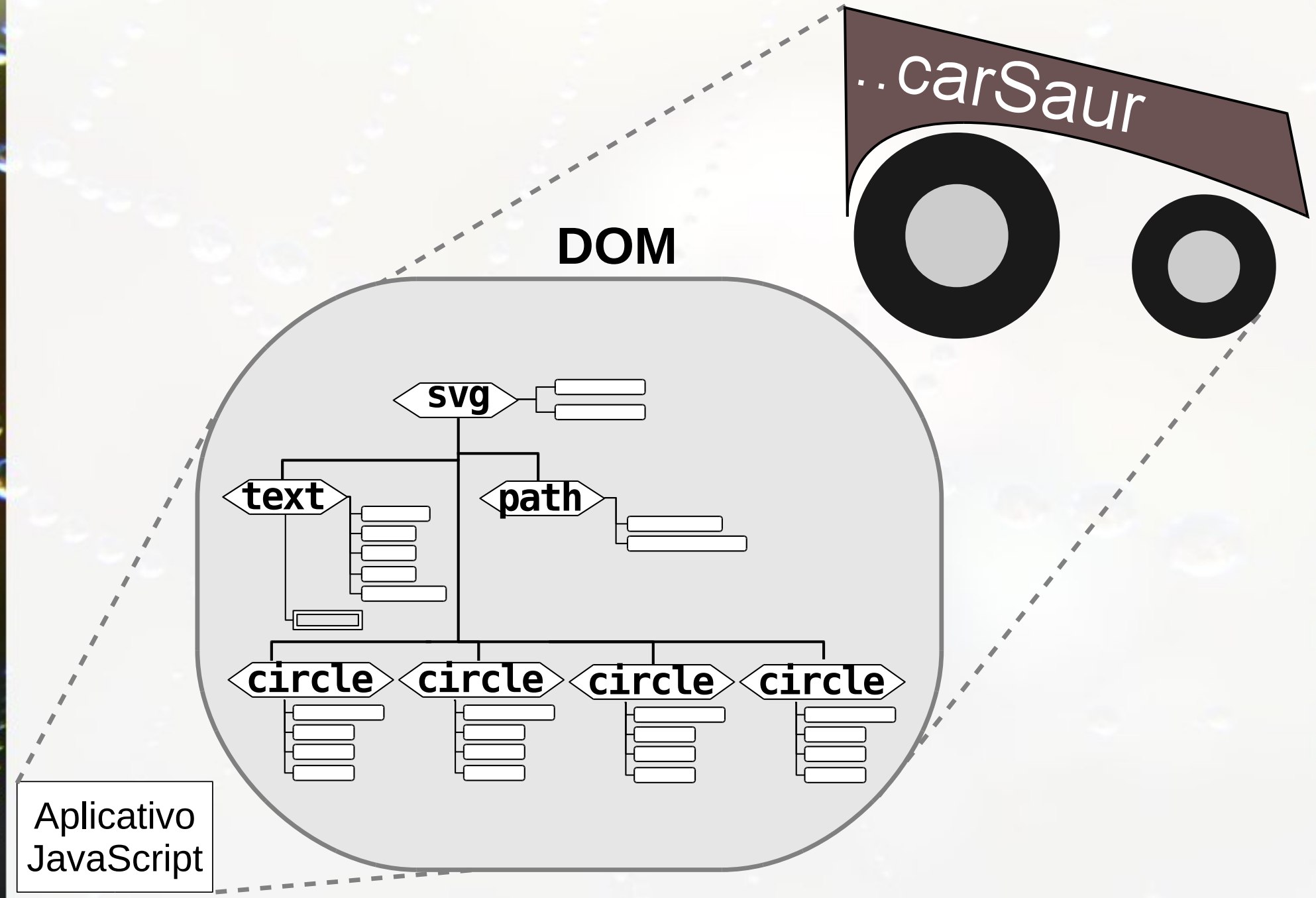
Primitiva	Descrição	Atributos	
<path>	Descreve um trajeto que usualmente será usado para a definição de contornos de polígonos.	style	Estilo de apresentação. Neste caso define a cor de preenchimento e do contorno e a espessura do contorno.
		d	Sequência de contorno formada por letras que representam primitivas de descrição do contorno e coordenadas.
<text>	Insere um texto.	style	Estilo de apresentação. Neste caso define a cor e fonte da letra.
		x, y	Coordenadas do esquerdo inferior do texto.

Carro SVG

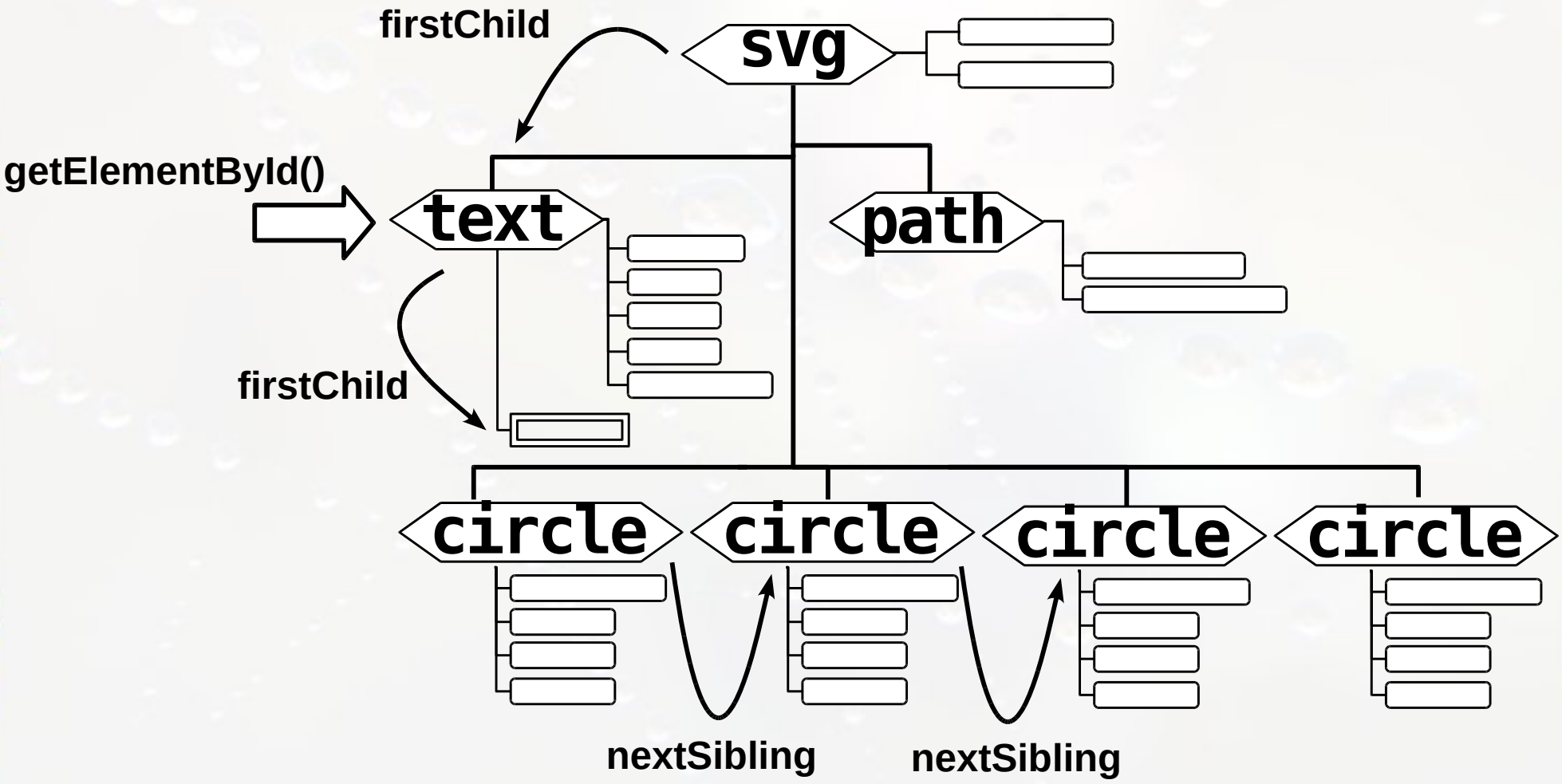
```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
     width="181" height="130">

  <circle style="fill:#1a1a1a" cx="44" cy="90" r="40" />
  <circle style="fill:#cccccc" cx="44" cy="90" r="20" />

  <circle style="fill:#1a1a1a" cx="140" cy="102" r="28" />
  <circle style="fill:#cccccc" cx="140" cy="102" r="14" />

  <path style="fill:#6c5353; stroke:#000000; stroke-width:1px"
        d="M 180.322,82.637687 172.30769,42.566127
0.50088787,1.4927774
           1.5026779,82.637677 c -2.50447,-82.14667965
           178.8193221,1e-5 178.8193221,1e-5 z" />

  <text style="fill:white; font-size:28px; font-family:Arial"
        x="9" y="30"
        transform="rotate(10)">
    ..carSaur
  </text>

</svg>
```

DOM em SVG


svg
width = “181”
height = “130”
text
class = “nameStyle”
x = “9”
y = “30”
r = “20”
transform = “rotate(10)”
..carSaur
..carSaur
path
class = “frame”
d = “M 180.322,...”
circle
class = “tyre”
cx = “44”
cy = “90”
r = “40”
circle
class = “rim”
cx = “44”
cy = “90”
r = “20”
circle
class = “tyre”
cx = “140”
cy = “102”
r = “28”
circle
class = “rim”
cx = “140”
cy = “102”
r = “14”

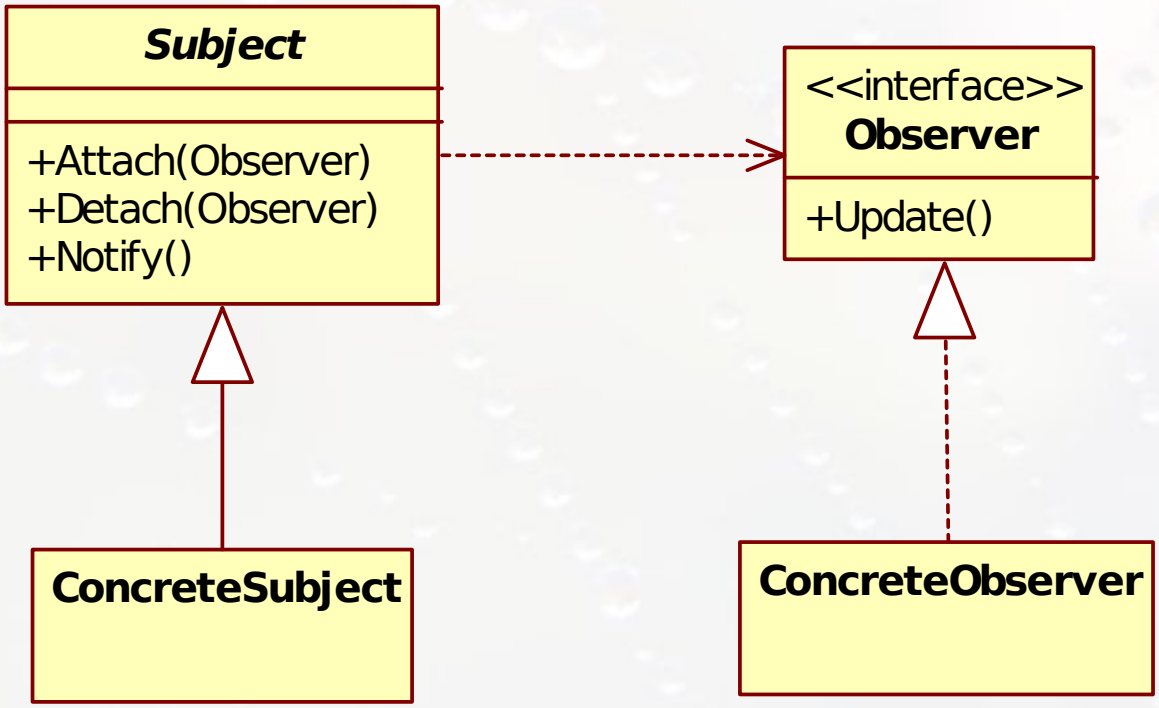

..carSaur
DOM
svg
text
path
circle
circle
circle
circle
Aplicativo
JavaScript

SVG em DOM


firstChild
svg
getElementById()
text
path
firstChild
circle
circle
circle
circle
nextSibling
nextSibling

Pattern Observer

Eventos
Pattern *Observer*



Eventos e *Pattern Observer*

Eventos e GUI

- Cada ação do usuário ao interagir com uma Interface produz um evento: arrastar o mouse, clicar em um botão, etc.
- Objetos podem ser notificados da ocorrência de um evento

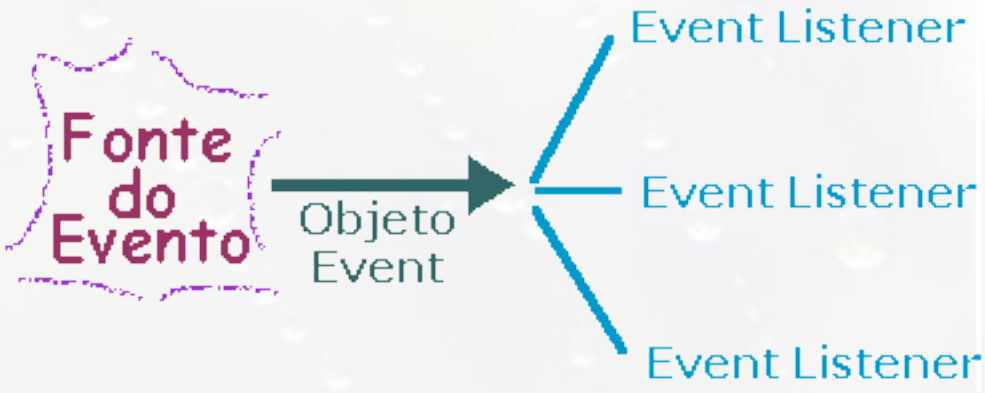
JavaBeans
Eventos

- Seguem o padrão *Observer*
- Registro de evento detectados automaticamente
- Registros de observadores (*listeners*) são “descobertos” por introspecção:
  - add<evento>Listener(<evento>Listener)
  - remove<evento>Listener(<evento>Listener)

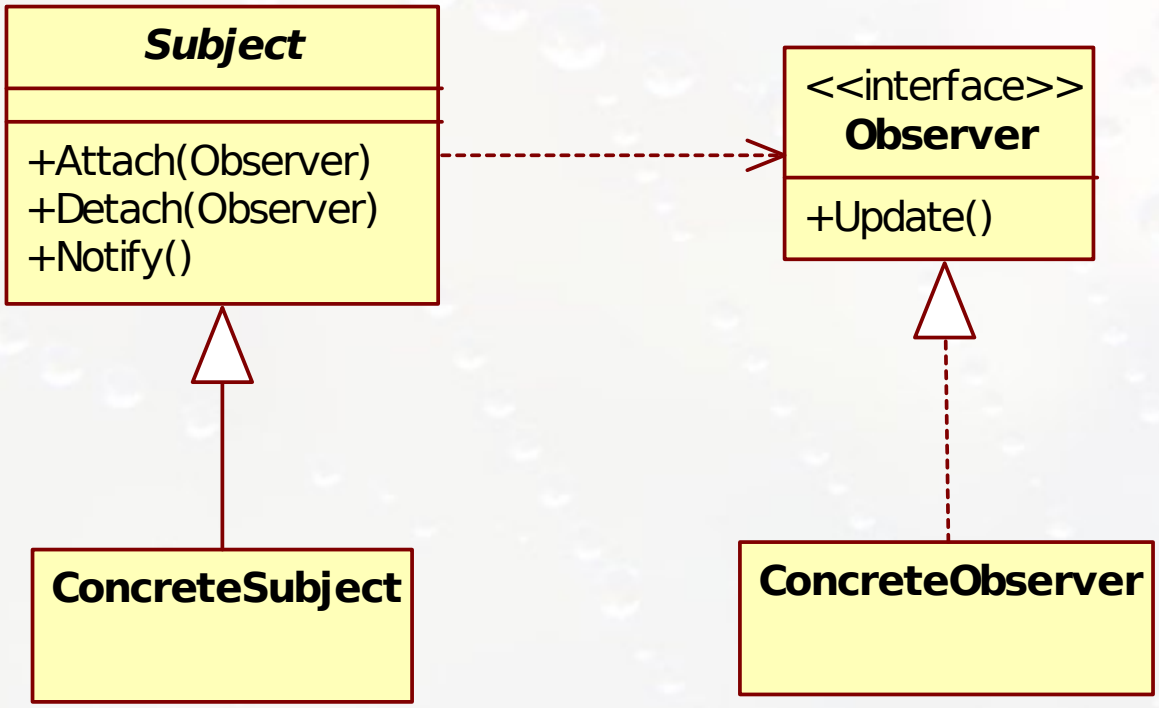
Eventos

Os eventos em Java são representados através de objetos.

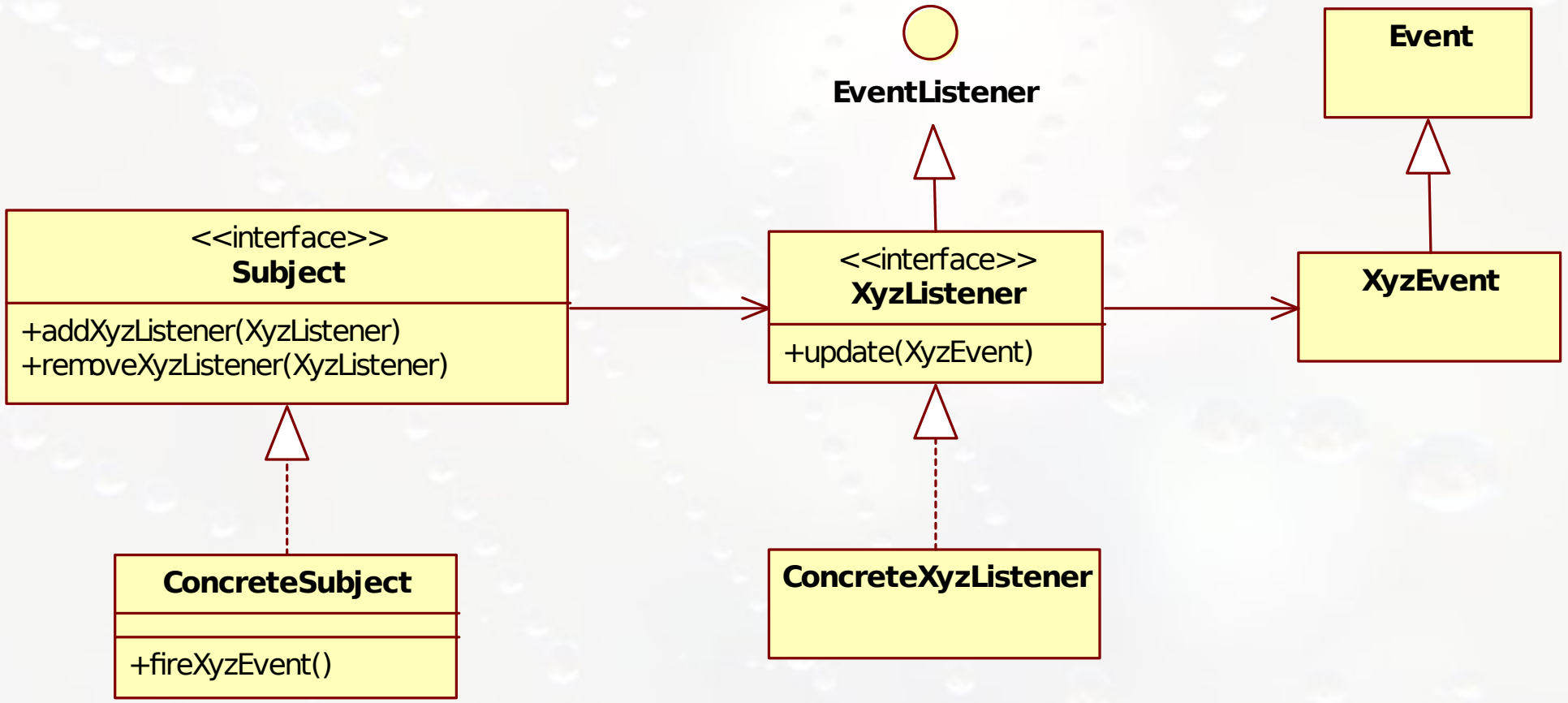
Tais objetos (eventos) podem ser capturados por objetos através de uma "escuta" (*listener*).



Eventos
Pattern *Observer*

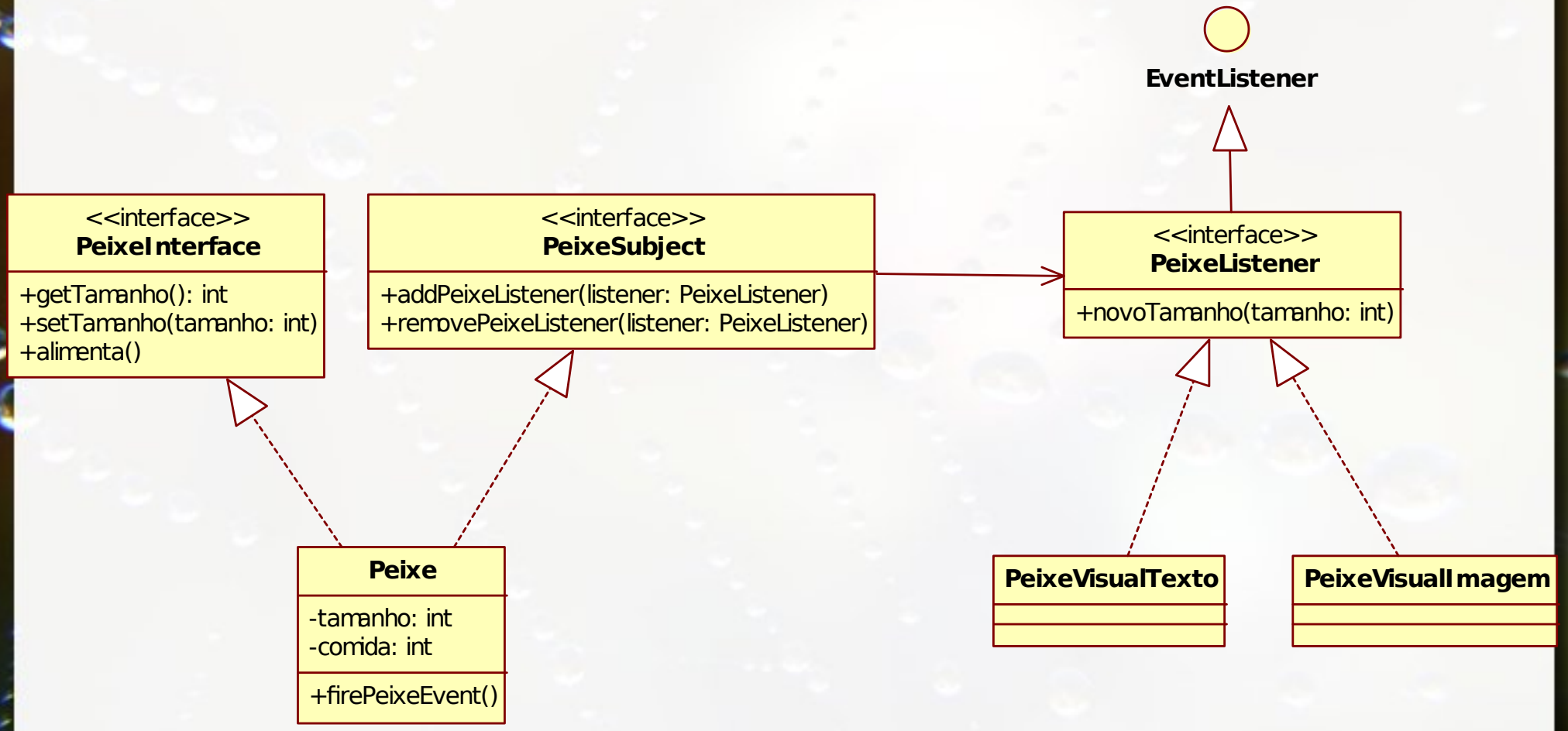


Eventos
Padrão *Listener*



Exemplo do Peixe

Exemplo do Peixe



Estilos Arquiteturais

Estilos de Controle

Baseada em Eventos

Event-driven systems

- Driven by externally generated events where the timing of the event is outwith the control of the sub-systems which process the event.
- Two principal event-driven models
  - Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so;
  - Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing.
- Other event driven models include spreadsheets and production systems.

Baseada em Eventos

- Componentes interagem através da difusão (broadcast) de eventos
- Ação inicia com um componente que 'anuncia' um evento
- Evento anunciado pode disparar operações em outros componentes

(Abowd, 1995)

- Exemplo: Publish-Subscribe

Eventos na Web

```
<html>
<head>
    <script type="text/javascript">
        function clicado()
        {
            alert("Clicou");
        }
    </script>
</head>

<body>
    <a href="#" onclick="clicado()">
        Clique aqui</a>
</body>
</html>
```

Estilos Arquiteturais

Estilos de Controle

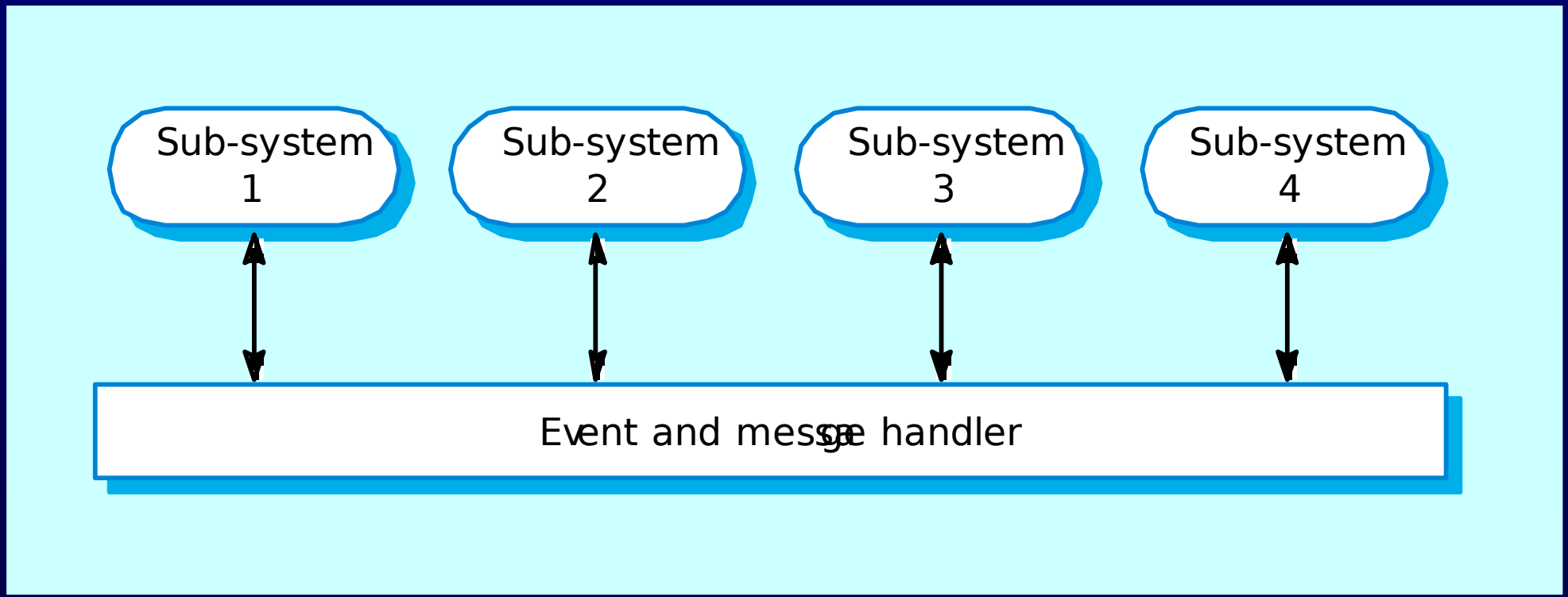
Baseado em Eventos

Broadcast

Broadcast model

- Effective in integrating sub-systems on different computers in a network.
- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event.
- Control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them.
- However, sub-systems don’t know if or when an event will be handled.

Selective broadcasting



Estilos Arquiteturais

Estilos de Controle

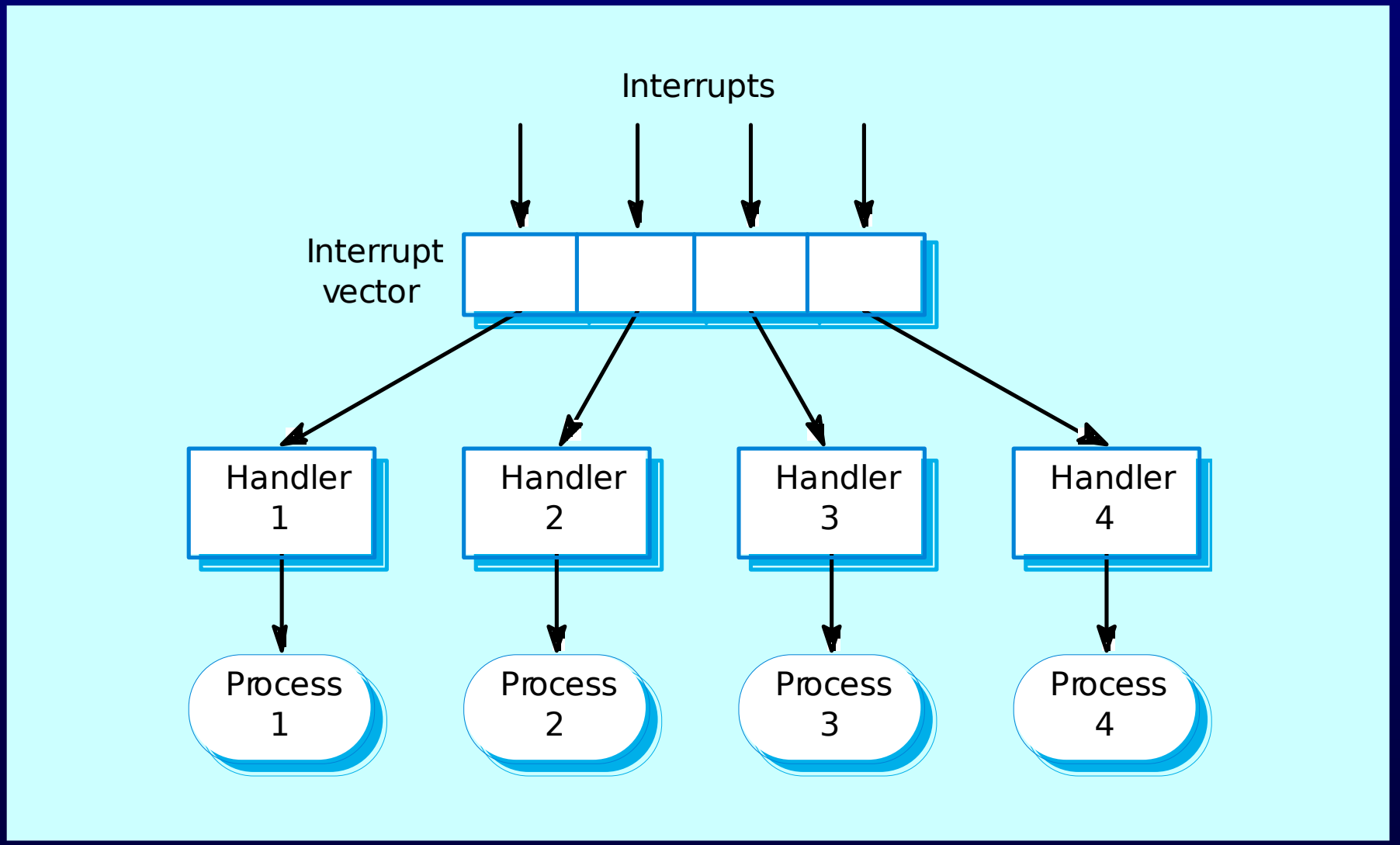
Baseado em Eventos

Interrupção

Interrupt-driven systems

- Used in real-time systems where fast response to an event is essential.
- There are known interrupt types with a handler defined for each type.
- Each type is associated with a memory location and a hardware switch causes transfer to its handler.
- Allows fast response but complex to program and difficult to validate.

Interrupt-driven control



Pattern Prototype

Pattern Prototype

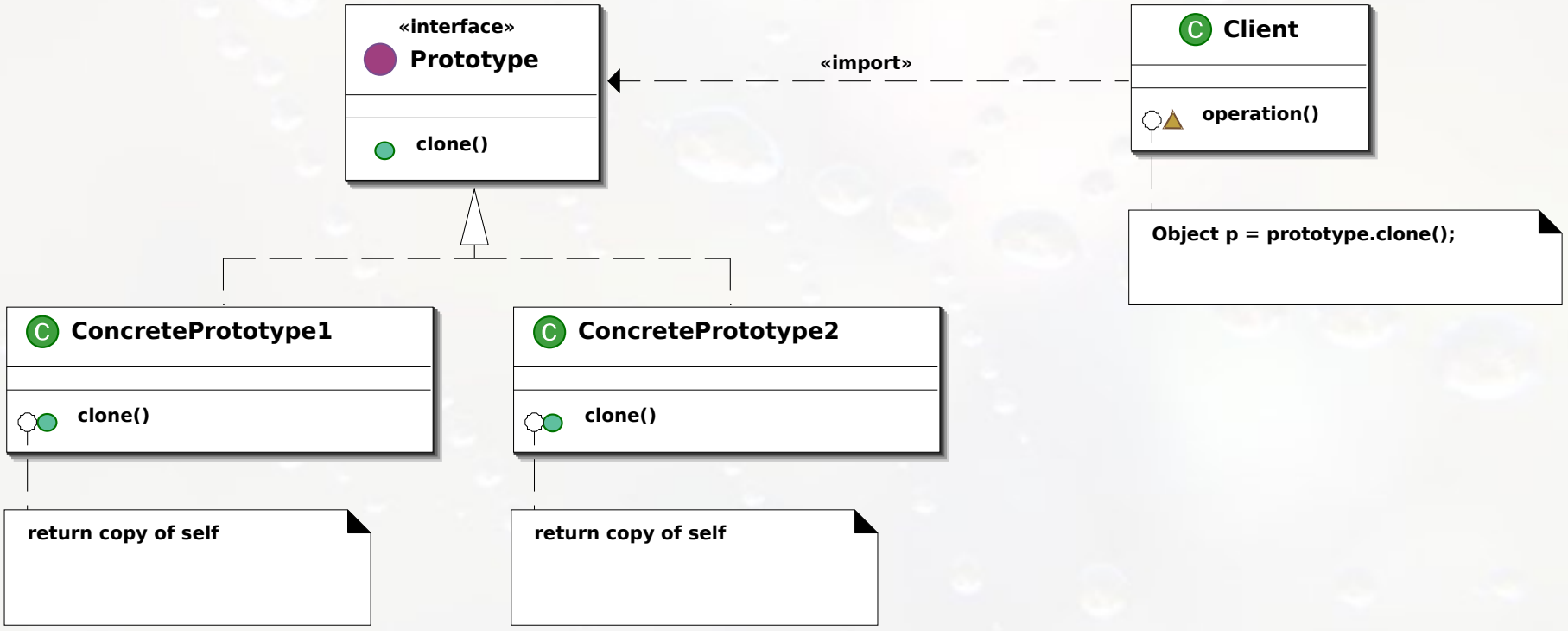
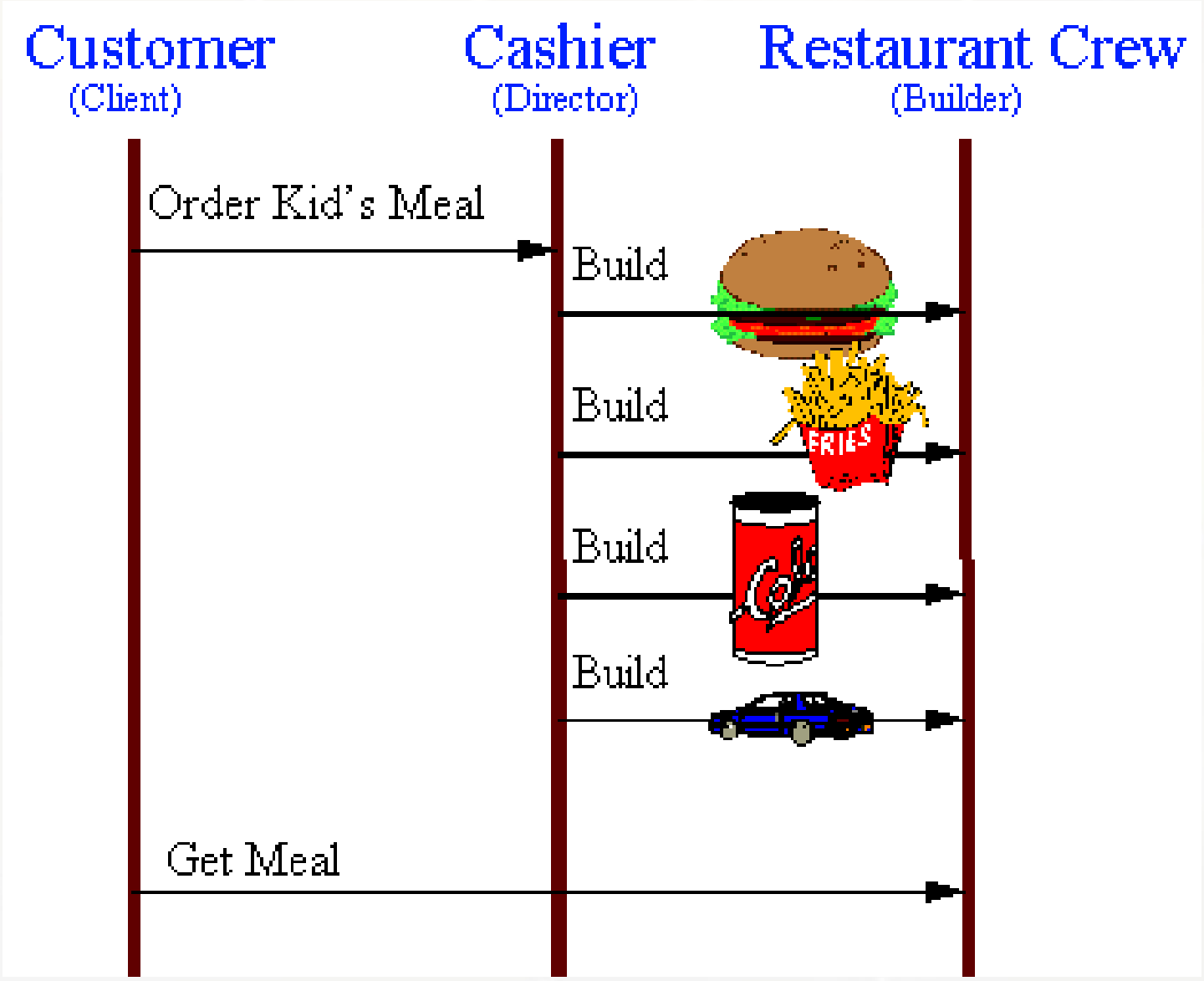


Imagem de Giacomo Ritucci (http://en.wikipedia.org/wiki/File:Prototype_UML.svg)

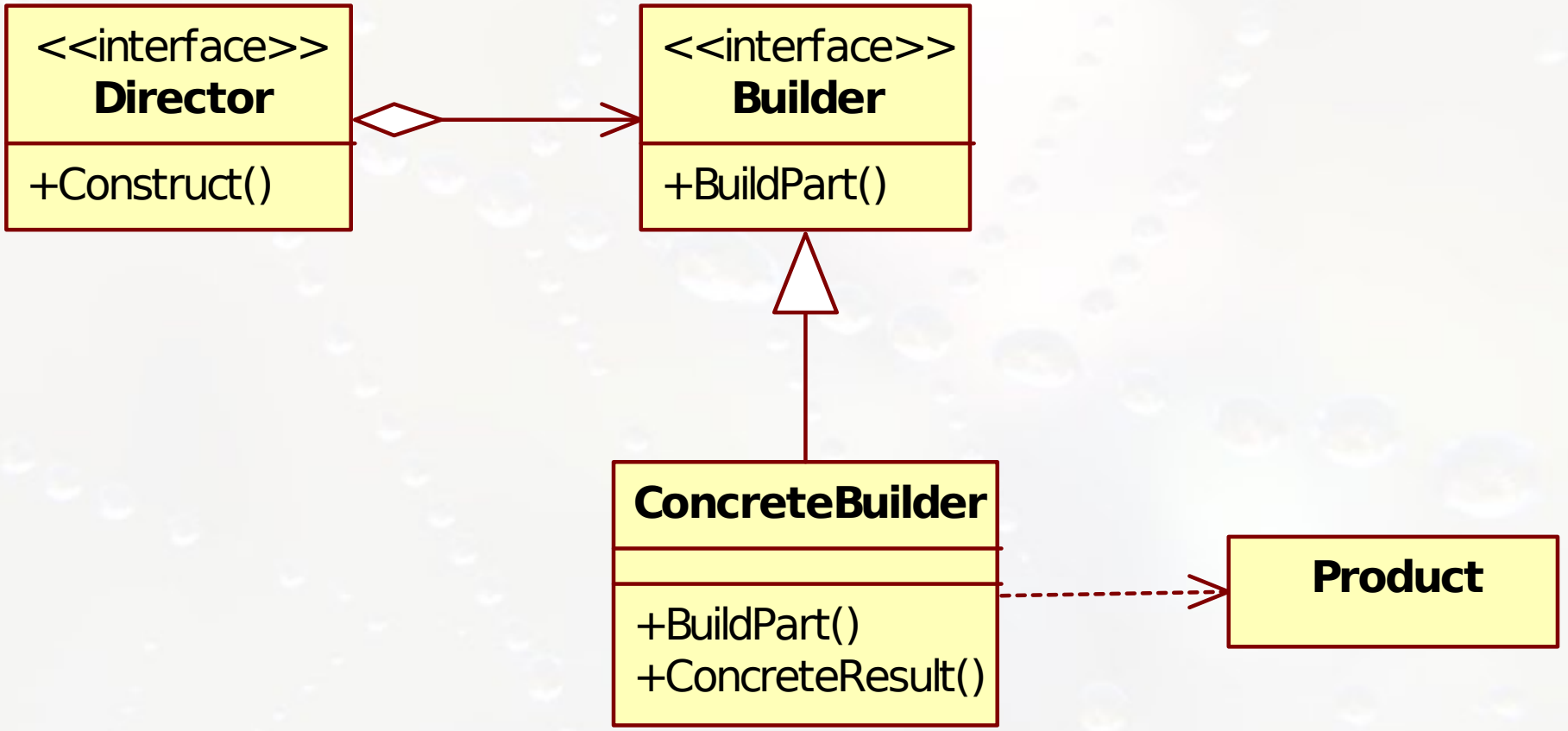
Pattern Builder

Builder

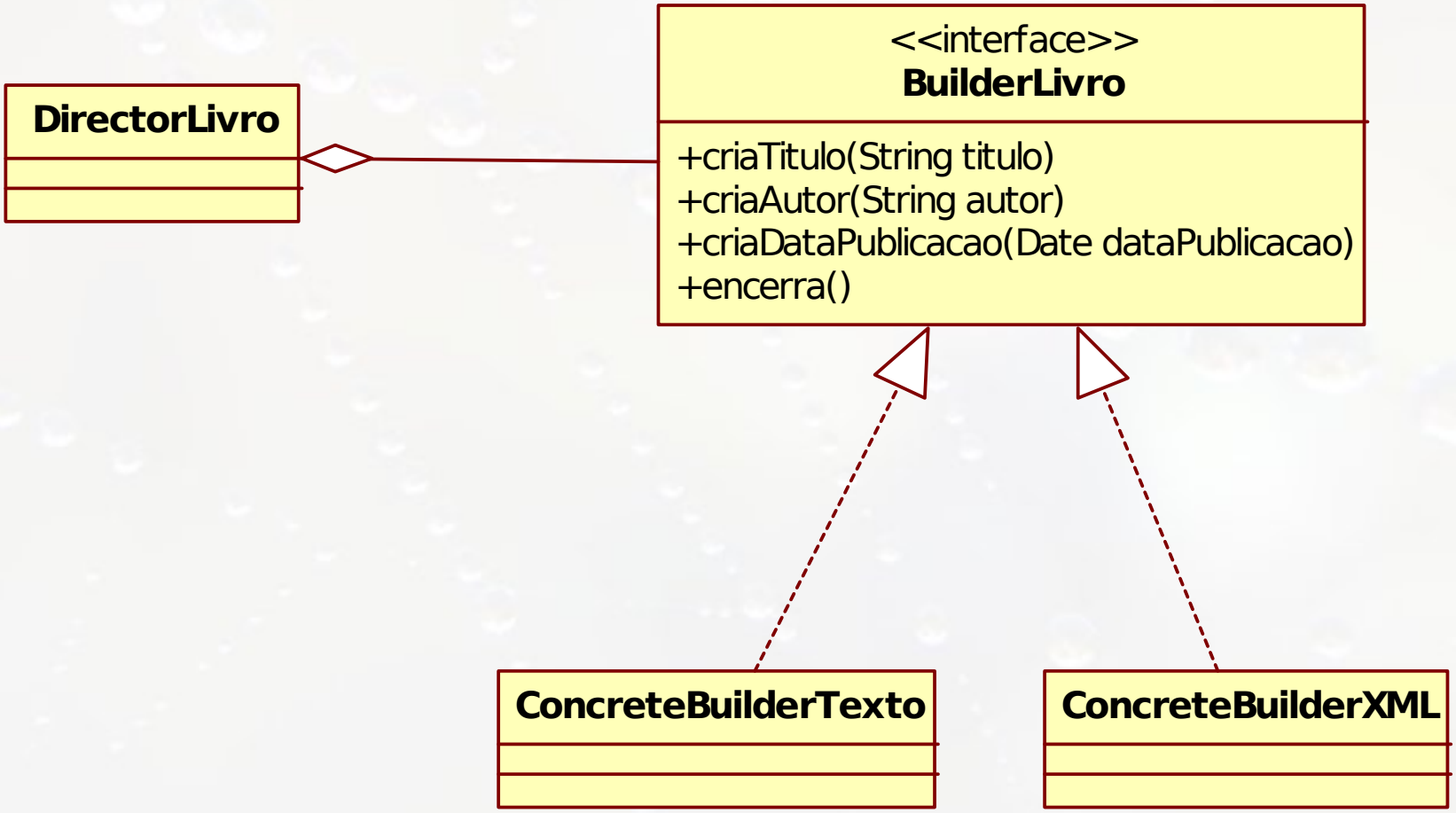


(AG Communication Systems, 1999)

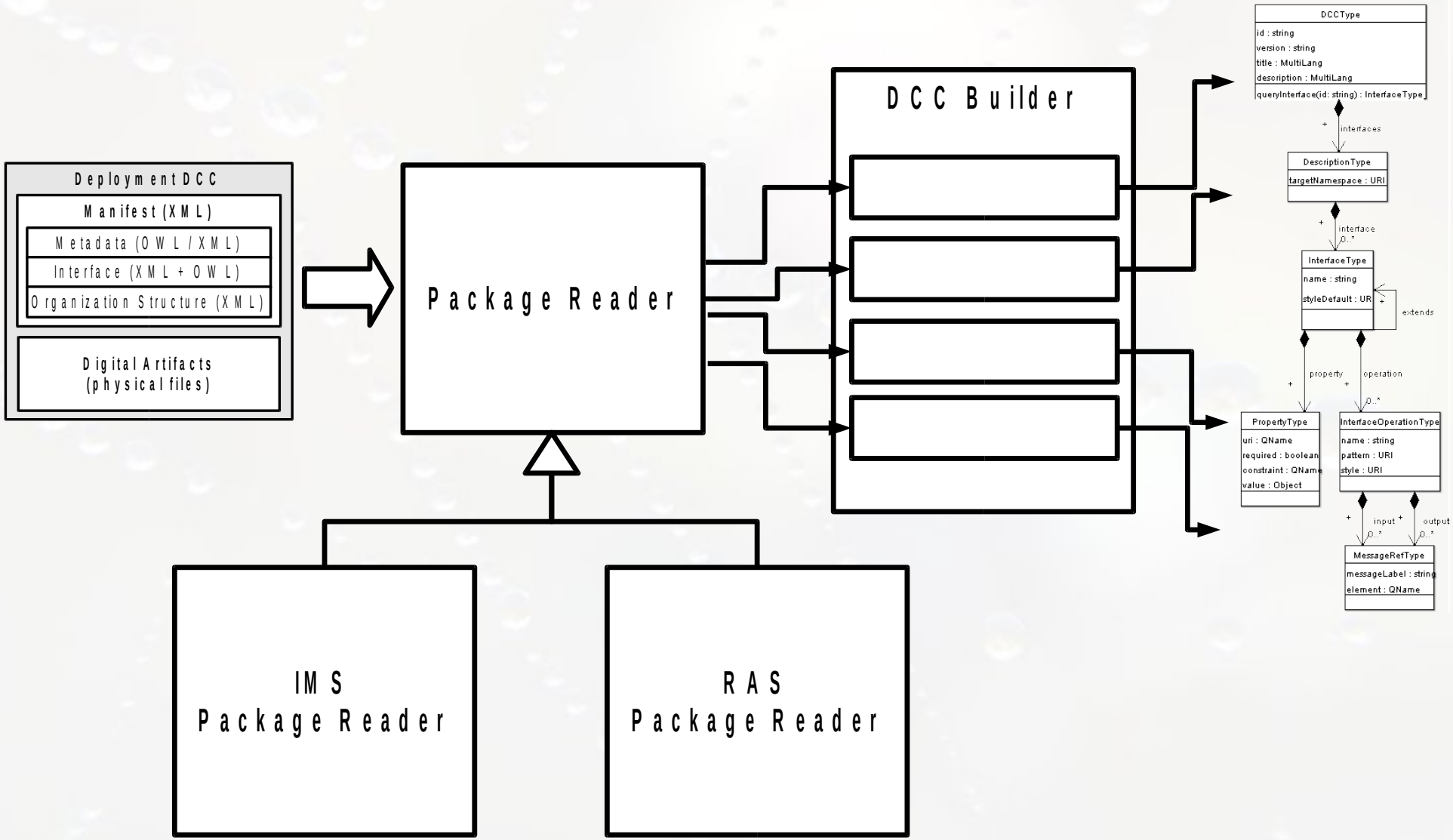
Pattern Builder



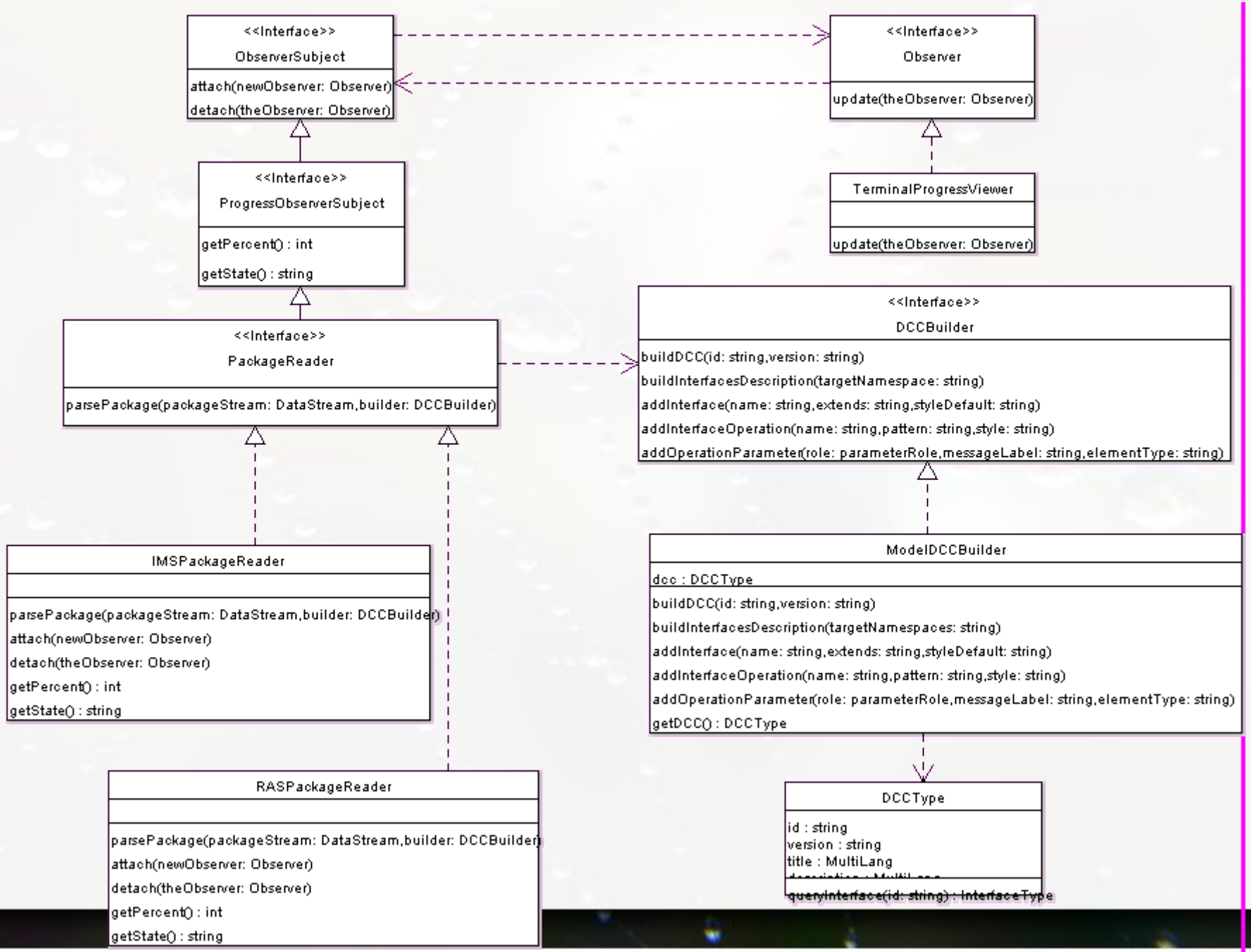
Pattern Builder
Livro



DCC Builder


Deployment DCC
Manifest (XML)
Metadata (OWL / XML)
Interface (XML + OWL)
Organization Structure (XML)
Digital Artifacts
(physical files)
Package Reader
DCC Builder
IMS
Package Reader
RAS
Package Reader
DCCType
id : string
version : string
title : MultiLang
description : MultiLang
queryInterface(id: string) : InterfaceType
interfaces
DescriptionType
targetNamespace : URI
interface
InterfaceType
name : string
styleDefault : UR
extends
property
operation
PropertyType
uri : QName
required : boolean
constraint : QName
value : Object
InterfaceOperationType
name : string
pattern : URI
style : URI
input
output
MessageRefType
messageLabel : string
element : QName

DCC Builder Model



SAX

- Tecnologia para acesso a documentos XML
- API baseada em eventos.
- Se tornou a mais estável API XML largamente utilizada [DOD01].
- Iniciou como uma solução para acesso a documentos XML por programas Java.
- Hoje tem sido portada para outras linguagens de programação, tal como: C++, Pascal, Perl, Phyton, etc.

SAX - Estudo de Caso

```
<FICHARIO>
  <INDIVIDUO nome="Asdrubal da Silva">
    <IDADE>15</IDADE>
    <SEXO>masculino</SEXO>
  </INDIVIDUO>
  <INDIVIDUO nome="Quincas Borba">
    <IDADE>33</IDADE>
    <SEXO>masculino</SEXO>
  </INDIVIDUO>
  <INDIVIDUO nome="Doriana Margarina">
    <IDADE>42</IDADE>
    <SEXO>feminino</SEXO>
  </INDIVIDUO>
</FICHARIO>
```

SAX - Estudo de Caso

```
========== Inicio do Documento ==========
Inicio de elemento: DOCUMENTO
  Inicio de elemento: INDIVIDUO
  (atributos): nome=Asdrubal da Silva;
    Inicio de elemento: IDADE
      Texto: 15
    Final de elemento : IDADE
    Inicio de elemento: SEXO
      Texto: masculino
    Final de elemento : SEXO
  Final de elemento : INDIVIDUO
  Inicio de elemento: INDIVIDUO
  (atributos): nome=Quincas Borba;
    Inicio de elemento: IDADE
      Texto: 33
    Final de elemento : IDADE
    Inicio de elemento: SEXO
      Texto: masculino
    Final de elemento : SEXO
  Final de elemento : INDIVIDUO
    Inicio de elemento: INDIVIDUO
      (atributos): nome=Doriana Margarina;
        Inicio de elemento: IDADE
          Texto: 42
        Final de elemento : IDADE
        Inicio de elemento: SEXO
          Texto: feminino
        Final de elemento : SEXO
      Final de elemento : INDIVIDUO
    Final de elemento : DOCUMENTO
    ========== Final do Documento ==========
```

Instanciar o *parser* SAX

```
SAXParserFactory spf =
                SAXParserFactory.newInstance();

SAXParser sp = spf.newSAXParser();
```

- A classe SAXParser representa o *parser* SAX.
- SAXParserFactory – fábrica de objetos SAXParser
  - define objetos capazes de construir objetos SAXParser

Objetos que manipulam os eventos

```
XMLReader xr = sp.getXMLReader();

xr.setContentHandler(this);

ErroSAX es = new ErroSAX();
xr.setErrorHandler(es);
```

- A própria classe (this) manipula eventos de conteúdo.
- Um objeto da classe ErroSAX manipula eventos de erro.

Eventos de conteúdo

```
public class SAXBasico extends
             org.xml.sax.helpers.DefaultHandler
{

  public void startDocument() ...

  public void startElement(...) ...

  public void characters (...) ...

  public void endElement(...) ...

  public void endDocument() ...

}
```

Eventos de conteúdo

Método	Acionado quando o *parser* encontra
`startDocument`	início do documento
`startElement`	início de um elemento
`characters`	conteúdo texto
`endElement`	final de um elemento
`endDocument`	final do documento

Iniciar processo de rastreamento

```
xr.parse("file:" + nomeArquivo);
```

- O método parse dispara todo o processo de rastreamento.
- A partir daí o documento XML será lido, analisado e os respectivos métodos serão notificados.

Referências

- AG Communication Systems. **Examples to Accompany: Design Patterns Elements of Reusable Object-Oriented Software**, 1999.
- Abowd, G. D., Allen, R., Garlan, D. **Formalizing style to understand descriptions of software architecture**. ACM Trans. Softw. Eng. Methodol., ACM Press, 1995, 4, 319-364.
- Alexander, Christopher; Ishikawa, Sara; Silverstein, Murray. **A Pattern Language: Towns, Buildings, Construction**. Oxford University Press, 1977.
- Krueger, C. W. **Software Reuse**. ACM Comput. Surv., ACM Press, 1992, 24, 131-183.
- Mcilroy, M. D. Naur, P. & Randell, B. (ed.) **Mass Produced Software Components**. Software Engineering: Report of a conference sponsored by the NATO Science Committee, 1968.

Referências

- Mili, H.; Mili, F. & Mili, A. **Reusing Software: Issues and Research Directions**. IEEE Transactions on Software Engineering, 1995, 21, 528-562.
- Shaw, M. **Abstraction Techniques in Modern Programming Languages**. IEEE Software, 1984, 1, 4, 10-26.
- Sommerville, I. (2007) **Software Engineering**, 8th. ed. Addison Wesley.

André Santanchè
http://www.ic.unicamp.br/~santanche

License

- These slides are shared under a Creative Commons License. Under the following conditions: Attribution, Noncommercial and Share Alike.
- See further details about this Creative Commons license at: http://creativecommons.org/licenses/by-nc-sa/3.0/