UnB - Universidade de Brasília
FGA - Faculdade do Gama
DAS - Desenvolvimento Avançado de Software

Trabalho Final

Frameworks / Desenvolvimento Baseado em Componentes (DBC)

Grupo:
Antônio Carvalho - 11/0058496
Flávio da Costa Paixão - 12/0117894
João Paulo Alves Ferreira - 10/0013929
Renata Francelino de Souza - 15/0045603
Vitor Nere Araújo Ribeiro - 13/0137413

1. ***O que é o framework Angular?***

É uma Plataforma e Framework JavaScript para desenvolvimento de aplicação em HTML e TypeScript. Angular em si é desenvolvimento em TypeScript, que implementa funcionalidades essenciais e opcionais como um conjunto de bibliotecas TypeScript no qual são importadas na aplicação. As bibliotecas provém uma enorme quantidade de funcionalidades que torna trivial o desenvolvimento de complexas ferramentas modernas da atualidade, como sincronização de dados, controle de rotas, e animações. Angular também provê uma série de convenções de como a aplicação deveria ser desenvolvida no qual beneficia times grandes que necessitam trabalhar juntos em um único código base, sendo um dos únicos Frameworks JavaScript a provê um guia de estilo compreensivo e com uma grande quantidade de guias de como você deveria escrever seu código com o Framework.

2. ***Qual seu propósito (desenvolvimento web, mobile, frontend)?***

Desenvolvido com o propósito de ser um Framework multiplataforma web e mobile para o desenvolvimento de aplicações em HTML e TypeScript baseada em componentes onde facilita a reutilização.

3. ***Como ele é estruturado (há diagrama de classes, arquitetural, dentre outros)?***

Angular é uma plataforma e framework para criação de aplicativos clientes em *HTML* e *TypeScript*. Escrito em *TypeScript*, ele implementa as funcionalidades principal e opcional como um conjunto de bibliotecas *TypeScript* que são importadas para as aplicações criadas.

Os blocos de construção básicos de um aplicativo Angular são *NgModules* , que fornecem um contexto de compilação para os componentes. Um aplicativo Angular portanto é definido por um conjunto de *NgModules* e eles, por sua vez, coletam códigos relacionados em conjuntos funcionais. Um aplicativo sempre tem pelo menos um módulo raiz que permite a autoinicialização e normalmente tem muito mais módulos de recursos .

Os componentes definem as *views* , que são conjuntos de elementos de tela que o Angular pode escolher e modificar de acordo com a lógica e os dados do programa. Todo aplicativo tem pelo menos um componente raiz.

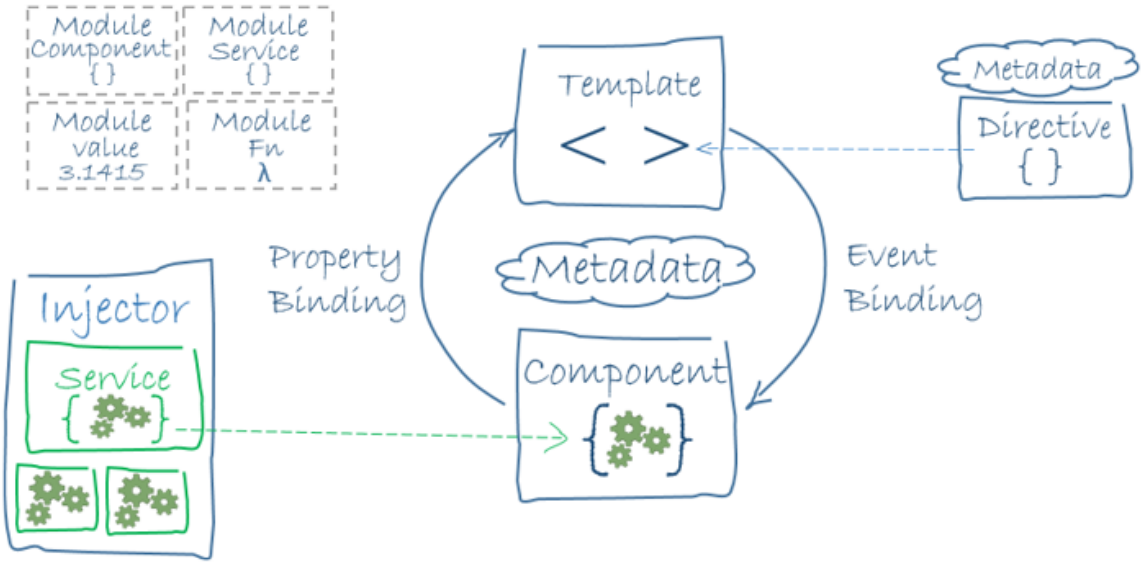
Componentes usam *services*, que fornecem funcionalidades específicas não diretamente relacionada às *views*. Os provedores de serviços podem ser injetados em componentes como dependências, tornando o código modular, reutilizável e eficiente.

Ambos os componentes e serviços são classes, com *decoradores* que definem seu tipo e fornece metadados que informam ao Angular como usá-los.

Os metadados de uma classe de componentes associam isso a um *template* que define uma exibição. Um *template* combina *HTML* comum com Angular *Directives* e marcação de ligação, que permitem que o Angular modifique o HTML antes de exibí-lo.

Os metadados de uma classe de serviço fornecem as informações que o Angular precisa para disponibilizá-los aos componentes por meio da injeção de *Dependency Injection (DI).*

Os componentes de um aplicativo geralmente definem várias exibições, organizadas hierarquicamente. Angular fornece o *Router Service* para ajudá-lo a definir caminhos de navegação entre visualizações. O *Router* oferece recursos sofisticados de navegação no navegador.



4. Quais são os pontos de extensão de funcionalidades desse framework (hot-spots)?

- *Ferramentas*
  - *Angular CLI - Oficial CLI (Interface de Linha de Comando) onde torna simples a criação de aplicações até a publicação. Já seguindo a folha de estilo*
  - *Angular Universal - API Oficial com objetivo de expandir e integrar os vários backends de um projeto desenvolvidos em outra linguagem como (NodeJS, ASP.NET, Java, etc.)*
  - *Augury - Ferramenta de extensão do Google Chrome para debugar.*

- *Bibliotecas de Dados*
  - *Angular Fire - Oficial biblioteca para Firebase*
  - *NgRx - Biblioteca de ferramentas assíncronas*
  - *Apollo - Gerenciador de dados desenvolvido com GraphQL*

- *Componentes de Interface de Usuário*
  - *Angular Material - Componentes do Material Design*
  - *NgxBootstrap - Components Bootstrap para Angular*
  - *PrimeFaces - Coleções de componentes de UI*

- *Desenvolvimento Multi-Plataforma*
  - *Ionic - Biblioteca de desenvolvimento de aplicações Mobile Webview multiplataforma com Angular e Codorva*
  - *NativeScript - Framework de desenvolvimento de aplicações Mobile Nativas com Angular e TypeScript*

5. ***Qual o modelo de componente definido para o framework? Ele está publicamente definido e acessível?***

5.1 Tipos de Componentes

- *Modules*
- *Components*
  - *Templates*
  - *Matadata*
  - *Data binding*
  - *Directives*
  - *Pipes*
- *Services*
- *Dependency Injection*

5.1 Formas de Interação

- Através de variáveis de entrada e saída entre componentes pais e filhos.
- Através de parâmetro de rotas.
- Através de serviços compartilhados entre os componentes

5.1 Definição de Recursos

Os recursos são definidos no @NgModule através de decoradores, onde o recurso fica disponível a todo componente ali importado.

6. ***O que é um componente de software para o framework adotado e como ele é estruturado?***

Um componente é uma "versão" simplificada de uma diretiva, que basicamente são marcadores ou extensões de elementos que compõem o DOM, estes marcadores informam ao Angular para inserir alguma funcionalidade específica a esse elemento.

Uma das principais vantagens do uso de componentes é criar trechos de códigos reaproveitados de maneira trivial, o que substitui a criação de diretivas com configurações mais complexas.

7. ***Como os detalhes de sua implementação são ocultados dos framework?***

Dentre os principais características do Angular, podemos destacar os componentes, templates, diretivas, roteamento, módulos, serviços, injeção de dependências e ferramentas de infraestrutura que automatizam tarefas, como a de executar os testes unitários de uma aplicação, para isso são utilizadas bibliotecas que servem para ocultar os detalhes implementação do framework.

8. ***Como suas interfaces de comunicação / composição devem ser definidas?***

- ***Module***
    - É o contêiner onde são declarados, importados e compartilhados os *components* para algum domínio específico da aplicação. É definido com um decorador de classe *@NgModule*. O *@NgModule* é uma função decoradora no qual recebe um objeto de metadados que descrevem as propriedades daquele módulo, são elas:
        - Declarations: Aqui são declarado os *components*, *directives* e *pipes* que pertencem a esse *@NgModule*
        - Exports: Aqui são exportados os *components* que deverão ficar visíveis e disponíveis para utilização por outros *components* de outro *@NgModules*
        - Imports: Aqui são importado os outros *@NgModule* no qual os *components* aqui declarados necessitam dos recursos
        - Providers: Aqui são declarado os *services* disponíveis aos *components* desse *@NgModule* e de todos *components* filhos. Sendo que a instância para todos os *components* filhos é única, permitindo a troca de informações.
        - Bootstrap: Aqui é declarado o principal *component* da aplicação no qual a aplicação será iniciada.

9. ***Como componentes de software são adaptados para o framework sob investigação? Glue-code? Herança? Adapter? Etc…***

O angular utiliza o conceito de herança entre os componentes, havendo um componente pai na qual envia ou recebe valores dos componentes filhos que podem realizar algum comportamento com as informações recebidas e enviar ao pai algum resultado processado.

Uma maneira de especificar no código a relação de herança, é na importação de um componente dentro de outro e utilizando o decorator @Input ou @Output para a troca de informações entre o componente importado e importador (filho e pai), é especificando um atributo do componente pai onde será possível enviar um valor @Input ou receber um valor @Output. O exemplo a seguir pode ser encontrado na documentação do Angular.

componente-interação / src / app / hero-child.component.ts

```
1. import { Component, Input } from '@angular/core';
2.
3. import { Hero } from './hero';
4.
5. @Component({
6.   selector: 'app-hero-child',
7.   template: `
8.     <h3>{{hero.name}} says:</h3>
9.     <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
10.  `
11. })
12. export class HeroChildComponent {
13.   @Input() hero: Hero;
14.   @Input('master') masterName: string;
15. }
```

O exemplo acima mostra o componente heroChildComponent herdando propriedades do componente Hero, conforme descrito na linha 13, e utilizando no template. Por meio dessa declaração de entrada, marcada pelo decorator @Input, é possível acessar propriedades de Hero.

component-interaction/src/app/hero-parent.component.ts

```
1. import { Component } from '@angular/core';
2.
3. import { HEROES } from './hero';
4.
5. @Component({
6.   selector: 'app-hero-parent',
7.   template: `
8.     <h2>{{master}} controls {{heroes.length}}
   heroes</h2>
9.     <app-hero-child *ngFor="let hero of heroes"
10.       [hero]="hero"
11.       [master]="master">
12.     </app-hero-child>
13.   `
14. })
15. export class HeroParentComponent {
16.   heroes = HEROES;
17.   master = 'Master';
18. }
```

O exemplo acima o HeroParentComponent utilizando o seletor do HeroChildComponent que foi declarado como um componente de entrada pelo @NgModule que ambos estão declarados. Mostra o HeroParentComponent enviando atributos ao HeroChildComponent .

10. ***Quais são os modos geralmente utilizados para composição componentes / frameworks?***

O modo utilizado para a composição de componentes no framework é única e definida pela arquitetura. Onde um @NgModule principal que pode ser comparado ao index de outras plataformas declara e importa os módulos e componentes de toda a aplicação. NgModules configura o injetor e o compilador e ajuda a organizar componentes, diretivas e canais do próprio módulo. Não necessariamente precisa declarar todos os componentes utilizados, mas no mínimo deve importar um outro @NgModule que declara esses componentes para que a aplicação esteja integrada como um todo. No Angular, as bibliotecas nativas herdam do NgModule.

11. Como se dá a comunicação entre componentes / framework?

A comunicação entre os componentes pode ser feita: Passando a referência de um componente para outro, através do componente pai, através de parâmetro de rotas, através de serviços e comunicação entre serviço.

Passando a referência de um componente para outro:

Essa solução deve ser usada quando os componentes tiverem dependência entre eles. Por exemplo, menu suspenso e alternância suspensa. Eles geralmente não podem existir sem o outro.

Por exemplo, na criação de um componente *side-bar-toggle* que vai ter um componente *side-bar* como *input* e ao clicar no botão de alternância para abrir e fechar o componente da *side-bar.*

Código:

```
app component

<app-side-bar-toggle [sideBar]="sideBar"></app-side-bar-toggle>
<app-side-bar #sideBar></app-side-bar>
```

app.component.html hosted with ❤ by GitHub view raw

```
@Component({
  selector: 'app-side-bar-toggle',
  templateUrl: './side-bar-toggle.component.html',
  styleUrls: ['./side-bar-toggle.component.css']
})
export class SideBarToggleComponent {

  @Input() sideBar: SideBarComponent;

  @HostListener('click')
  click() {
    this.sideBar.toggle();
  }

}
```

side-bar-toggle.component.ts hosted with ❤ by GitHub view raw

```
@Component({
  selector: 'app-side-bar',
  templateUrl: './side-bar.component.html',
  styleUrls: ['./side-bar.component.css']
})
export class SideBarComponent {

  @HostBinding('class.is-open')
  isOpen = false;

  toggle() {
    this.isOpen = !this.isOpen;
  }

}
```

side-bar.component.ts hosted with ❤ by GitHub view raw

Comunicação através do componente pai:

Pode ser usado quando é fácil controlar o estado compartilhado entre os componentes por meio do componente pai e você não deseja criar um novo serviço ou criar um código clichê, devido a uma variável.

A implementação dessa abordagem é quase igual à passar a referência de um componente para outro, no entanto, a alternância de barra lateral não recebe componente da barra lateral. Em vez disso, o componente pai contém a propriedade sideBarIsOpened que é passada para o componente da barra lateral.
Código:

```
<app-side-bar-toggle (toggle)="toggleSideBar()"></app-side-bar-toggle>
<app-side-bar [isOpen]="sideBarIsOpened"></app-side-bar>
```

app.component.html hosted with ❤ by GitHub view raw

```
@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent {
  sideBarIsOpened = false;

  toggleSideBar(shouldOpen: boolean) {
    this.sideBarIsOpened = !this.sideBarIsOpened;
  }
}
```

app.component.ts hosted with ❤ by GitHub view raw

```ts
@Component({
  selector: 'app-side-bar-toggle',
  templateUrl: './side-bar-toggle.component.html',
  styleUrls: ['./side-bar-toggle.component.css']
})
export class SideBarToggleComponent {

  @Output() toggle: EventEmitter<null> = new EventEmitter();

  @HostListener('click')
  click() {
    this.toggle.emit();
  }

}
```

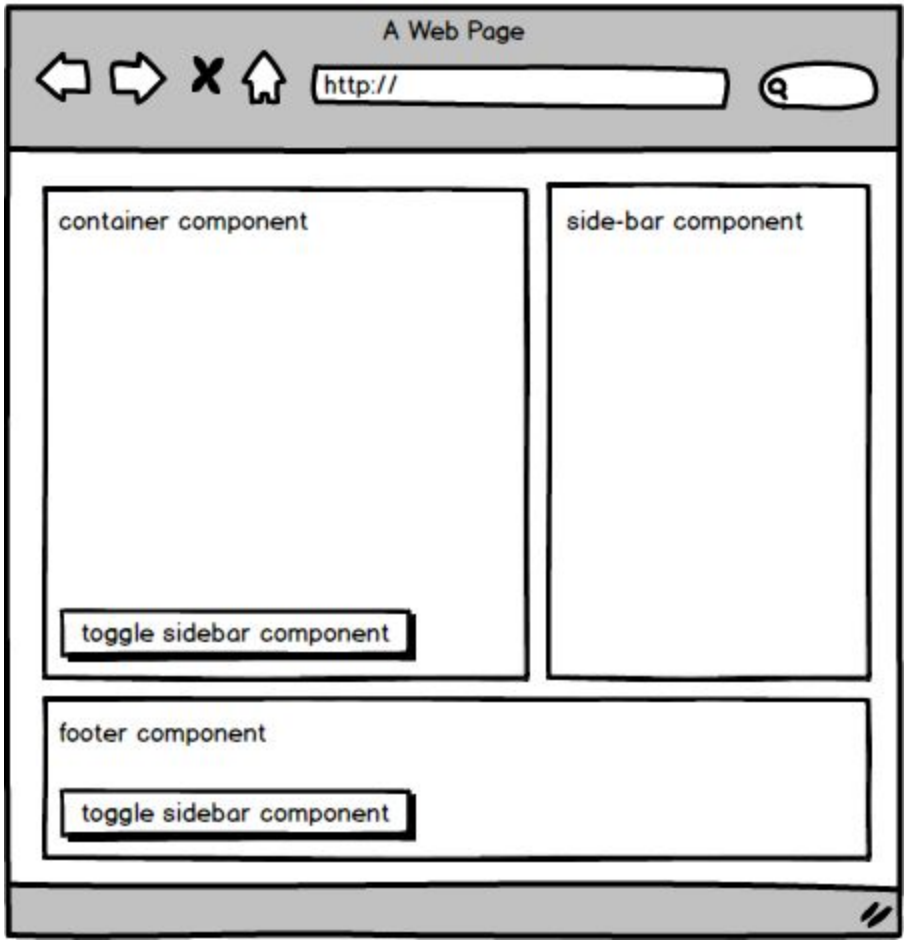
side-bar-toggle.component.ts hosted with ❤ by GitHub view raw

```ts
@Component({
  selector: 'app-side-bar',
  templateUrl: './side-bar.component.html',
  styleUrls: ['./side-bar.component.css']
})
export class SideBarComponent {

  @HostBinding('class.is-open') @Input()
  isOpen = false;

}
```

side-bar.component.ts hosted with ❤ by GitHub view raw

Comunicação através de serviço:

Finalmente esta opção é útil e deve ser usada quando quando se tem um componente que é controlado ou seu estado é solicitado a partir de múltiplas instâncias.



Agora com vários locais em todo o aplicativo que precisarão acessar nosso componente de barra lateral. Ao criar o side-bar.service.ts, assim teremos:

| side-bar.service.ts
| side-bar.component.ts
| Side-bar.component.html

Os serviços da barra lateral terão o método de alternar e alterar o evento, de modo que cada componente que injetará esse serviço possa ser notificado de que o painel foi aberto ou pode alterná-lo.

Neste exemplo, nenhum componente de barra lateral ou componente de barra lateral possui parâmetros de entrada, porque eles se comunicam por meio de serviço.

Código:

```
<app-side-bar-toggle></app-side-bar-toggle>
<app-side-bar></app-side-bar>
```

app.component.html hosted with ❤ by GitHub view raw

```
@Component({
  selector: 'app-side-bar-toggle',
  templateUrl: './side-bar-toggle.component.html',
  styleUrls: ['./side-bar-toggle.component.css']
})
export class SideBarToggleComponent {

  constructor(
    private sideBarService: SideBarService
  ) { }

  @HostListener('click')
  click() {
    this.sideBarService.toggle();
  }
}
```

side-bar-toggle.component.ts hosted with ❤ by GitHub view raw

```
@Component({
  selector: 'app-side-bar',
  templateUrl: './side-bar.component.html',
  styleUrls: ['./side-bar.component.css']
})
export class SideBarComponent {

  @HostBinding('class.is-open')
  isOpen = false;

  constructor(
    private sideBarService: SideBarService
  ) { }

  ngOnInit() {
    this.sideBarService.change.subscribe(isOpen => {
      this.isOpen = isOpen;
    });
  }
}
```

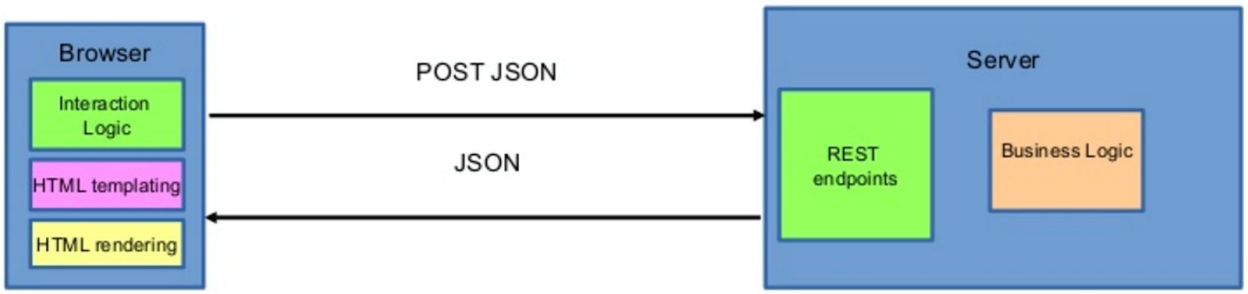
side-bar.component.ts hosted with ❤ by GitHub view raw

```typescript
@Injectable()
export class SideBarService {

  isOpen = false;

  @Output() change: EventEmitter<boolean> = new EventEmitter();

  toggle() {
    this.isOpen = !this.isOpen;
    this.change.emit(this.isOpen);
  }

}
```

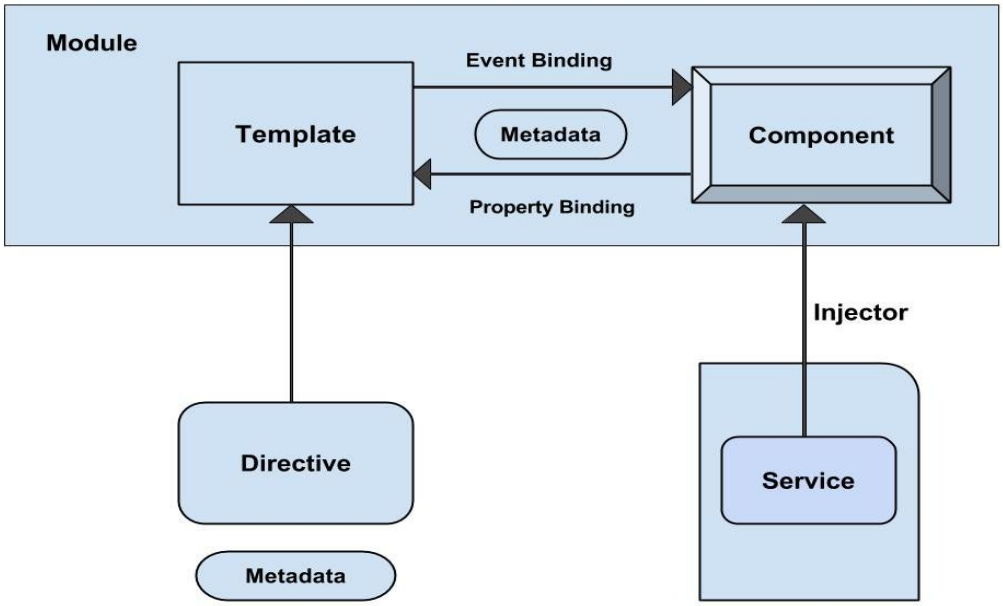
side-bar.service.ts hosted with ❤ by GitHub view raw

12. Há um diagrama comportamental (sequência, colaboração, etc) que demonstre tal comunicação?

Comunicação externa:



Comunicação interna:



Referências:

3 maneiras de se comunicar entre componentes angulares. *Disponível em:* <https://medium.com/dailyjs/3-ways-to-communicate-between-angular-components-a1e3f3304ecb> Acessado em: 27/06/2018

Curso - O que é Angular?.
Disponível em: <*https://www.devmedia.com.br/view/viewaula.php?idcomp=37620*>.
Acessado em: 27/06/2018

Architecture Overview*. Disponível em: <https://angular.io/guide/architecture>. Acessado em:* 27/06/2018

Aprenda Angular O Caminho Certo. *Disponível em: <https://www.sitepoint.com/angular-introduction/>. Acessado em:* 27/06/2018

Why Is React More Popular Than Angular? *Disponível em: <https://javascriptreport.com/why-is-react-more-popular-than-angular/ >. Acessado em:* 27/06/2018

What is Angular*. Disponível em: <https://angular.io/docs>. Acessado em:* 27/06/2018

What is Angular*. Disponível em: <https://developer.telerik.com/topics/web-development/what-is-angular/***> .***Acessado em:* 27/06/2018

SAPGNOLI, Luciana de Araújo. BECKER, Karin**.Um estudo sobre o Desenvolvimento Baseado em Componentes.** Programa de Pós Graduação em Ciência da Computação - PPGCC-PUCRS. Maio, 2003.