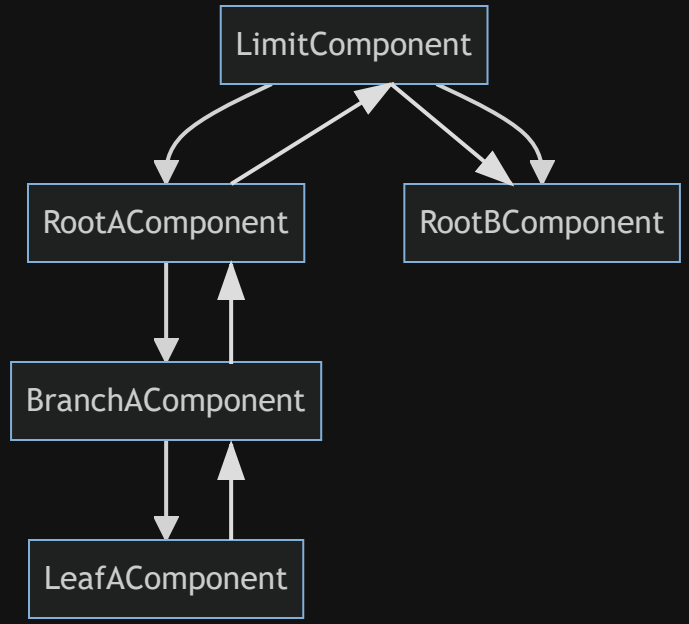
Limites

- Considérons l'arborescence suivante :



- Dans cet exemple, je veux qu'une modification sur le compteur depuis le composant LeafA soit répercutée sur le composant RootB
- Il n'y a pas de remontée automatique des évènements avec les @Output, il faut donc que le chaque composant s'occupe de faire remonter l'évènement à son parent, jusqu'a la racine
- Puis le composant racine fait redescendre la modification avec un @Input dans le composant B
- Un cas d'usage relativement simple pourtant déjà complexe à gérer

Services et injection de dépendances

Injection de dépendances

- L’injection de dépendances est un des principes fondamentaux d’Angular, qui mets en relation des consumers (qui ont besion d’une ressource) et des providers (qui fournissent une ressource)
- Les providers sont des services
- Les consumers peuvent être des composants, pipes, directives ou d’autres services
- L’injecteur gère une liste de providers, sous forme de singletons, et les fournit aux consumers qui en ont besoin

Services

- Un service est simplement une classe, annotée avec le décorateur @Injectable()

```
@Injectable({
  providedIn: 'root'
})
class MyService {
  ...
}
```

Utilisaton

- Dans le constructeur

```
constructor(public myService: MyService) { ... }
```

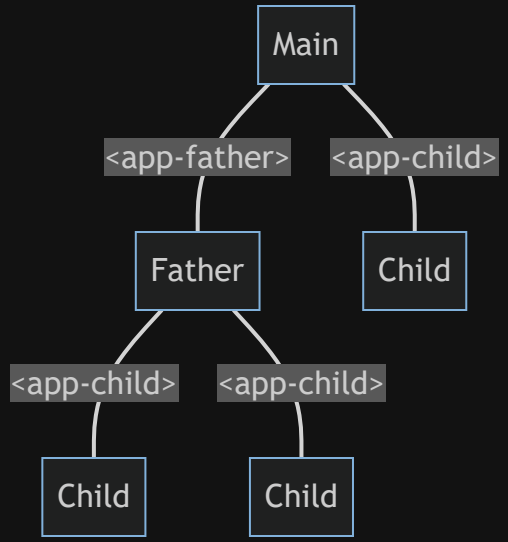
- Depuis Angular 14, avec la fonction inject()

```
myService = inject(MyService)
```

- Notre composant peut maintenant utiliser le service
- Pas besoin d’instancier myService, c’est l’injecteur qui s’en occupe pour nous

Définition des providers

- Pour la suite, considérons l'arborescence de composants suivante :



- Un service CounterService permet de stocker une valeur numérique, et de l'incrémenter
- Le composant father à une dépendances vers ce service, et le composant child deux

Définition des providers

- Une application Angular comporte plusieurs injecteurs, avec chaqun une liste de providers
  - root : unique pour l'application*
  - ElementInjector : spécifique pour un composant (ou une directive)

- *En réalité, l'injecteur root est un EnvironmentInjector (anciennement ModuleInjector), et il y a plusieurs EnvironmentInjector dans une application Angular

root

- On peut déclarer un service dans l'injecteur root dans les métadonnées de l'annotation @Injectable

```
@Injectable({
  providedIn: 'root'
})
export class CounterService { ... }
```

- On peut également les définir dans les providers des modules

```
@NgModule({
  ...
  providers: [CounterService],
)}
export class AppModule { }
```

- Pour une application standalone, on peut le définir dans bootstrapApplication()

```
bootstrapApplication(RootComponent, {
    providers: [ CounterService ]
}).catch(err => console.error(err));
```

Commentaires

- Pour les services que l'on crée, et que l'on souhaite inclure dans l'injecteur root, il est préferable de le déclarer dans le décorateur (avec providedIn: 'root').
- Tous les providers de vos modules importés se revtrouvent dans l'injecteur root (sauf dans le cas du lazy-loading), les providers d'un modules ne sont donc pas limités aux composants de votre module

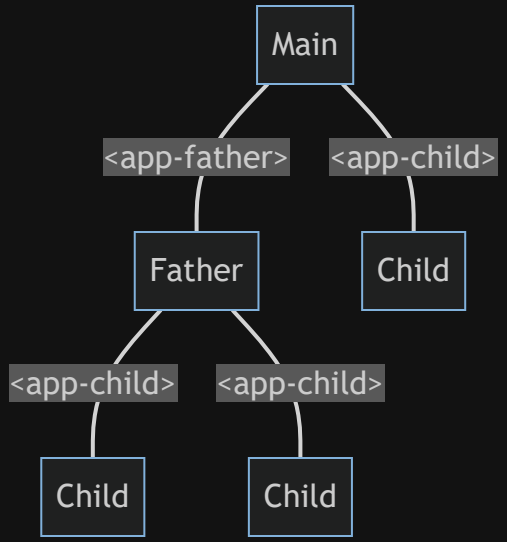
Element Injector

- Les composants et les directives ont également leur propre injecteur

```
@Component({
  providers: [CounterService],
...
```

- Dans ce cas, l'instance de CounterService est différente pour chaque composant
- L'injecteur est également accessible par les éléments fils
- Angular met à disposition des providers par défaut dans l'Element Injector, par exemple l'élément du DOM du composant, ou l'instance du composant lui même

Priorité de résolution



- Lorsque un élément doit résoudre une dépendance, il cherche d'abord dans son propre injecteur
- Puis il remonte les injecteurs parents un à un jusqu'a tomber sur l'élément racine
- A la fin, il demande à l'injecteur root
- Si la dépendance n'est toujours pas résolue, une exception est levée