EVOLUÇÃO CONTROLADA DE ARQUITECTURAS DE SERVIÇOS WEB

João Campinhos, João Costa Seco, and Jácome Cunha
NOVA LINCS, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa

Resumo

As arquitecturas de aplicações baseadas em micro serviços, ou aplicações baseadas em serviços web têm tido um crescimento significativo, muito pelo exponencial aumento da utilização de dispositivos móveis, cujas aplicações dependem quase totalmente deste tipo de interfaces. A evolução destes serviços constitui um problema de engenharia de software com impacto em, potencialmente, muitos milhões de utilizadores. Qualquer modificação ao comportamento ou à estrutura de uma interface remota pode causar erros de execução. Para resolver este problema que afecta software dependente de serviços de terceiros, criámos um modelo de programação que permite uma evolução segura de uma interface de servicos web, minimizando a introdução de incompatibilidades que poderiam eventualmente gerar um comportamento inesperado. Lidamos com a variabilidade no software de forma a representar simultaneamente várias versões ou variantes da mesma interface e do seu código de forma correcta e segura. Apresentamos o modelo de uma linguagem de programação, que garante que as alterações feitas ao código e respectiva interface não afectam negativamente a utilização corrente do sistema em causa.

Abordagem

- Permitir a execução de múltiplas versões em simultâneo.
- Assegurar a verificação de tipos entre versões
- Assegurar actualizações do cliente seguras e automáticas
- Utilização modular compatível com sistemas de versionamento representados em árvore

Servidor

- Código anotato com versões
- Routing em tempo de execução de acordo com a versão dada

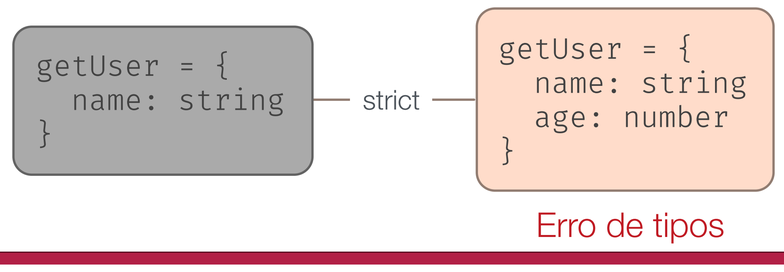
```
Version 1.0   { getUser() ⇒ { name: “John Doe” } }
Version 1.1-a { getUser() ⇒ { name: “John Doe” } }
Version 1.2-a { getUser() ⇒ { name: “John Doe”, age: 42 } }
Version 2.0-b { getUser() ⇒ { name:
  { first: “John”, last: “Doe” }, age: 42
}
```

Cliente

```
GET /getUser/ version=1.0 mode=strict  ⟹
  version=1.1-a { name: “John Doe” }
GET /getUser/ version=1.0 mode=subtype ⟹
  version=1.2-a { name: “John Doe”, age: 42 }
GET /getUser/ version=@2.0-b ⟹
  version=γ { name: { first: “John”, last: “Doe” }, age: 42 }
```

Sistema de Tipos

- Verificação de tipos através de anotações utilizadas pelo FlowType[1]
- O sistema de tipos verifica para cada versão se o tipo de retorno viola ou não o nível de compatibilidade das versões
- Posteriormente é utilizado o flowtype para a verificação estática do programa, com a versões já verificadas



Problema

Aplicações baseadas em serviços web (web mobile) estão no núcleo da internet

- Serviços Business to Developer
- Aplicações Multiplataforma
- Internet das Coisas
- APIs Públicas

Evolução das APIs é independente da evolução dos clientes

- Requer mais cuidado (compatibilidade com versões anteriores)
- Ainda assim pode causar erros de execução

Versão 1.0

```
getUser() ⇒ { name: “John Doe” } }
```

Versão 2.0

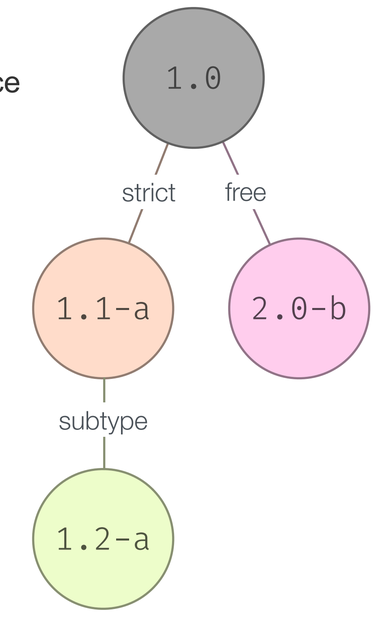
```
getUser() ⇒ { name: { first: “John”, last: “Doe” } }
```

Código Cliente Versão 1

```
getUser().name.toUpperCase() ← erro na versão 2
```

Sistema de versionamento

- Sistema de versionamento representado em árvore
- Cada nó corresponde a uma versão que contém a interface pública do serviço.
- Cada ramo da relação corresponde ao nível de compatibilidade entre versões.



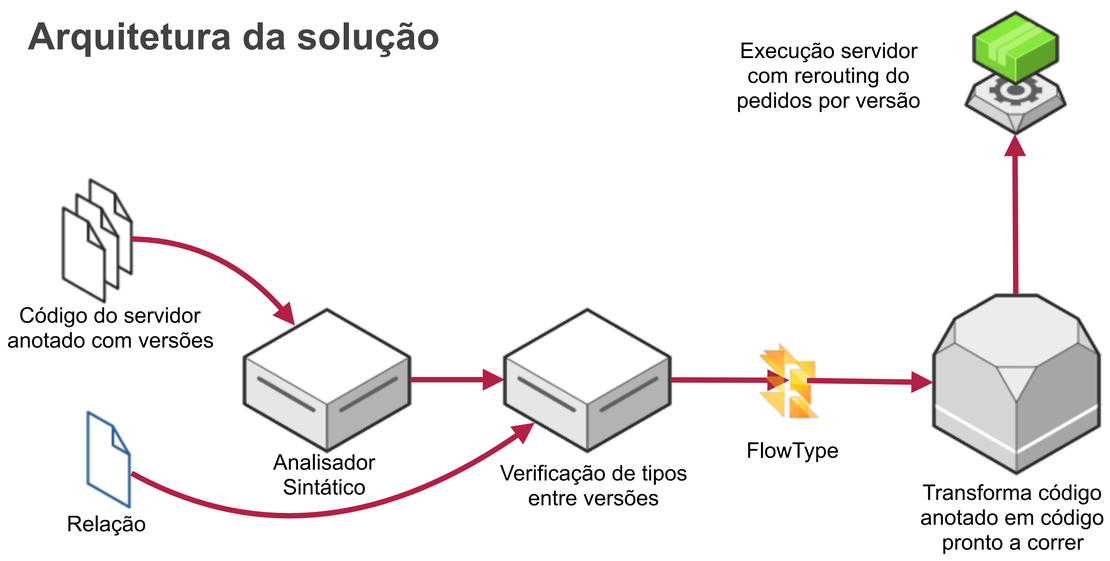
Três níveis de compatibilidade:

strict: os tipos entre versões têm de se manter iguais.

subtype: o tipo da versão mais recente pode ser subtipo da anterior.

free: não existem quaisquer restrições de tipos entre versões e não há garantias de compatibilidade.

Arquitetura da solução



Referências

1. A. Chaudhuri. Flow: a static type checker for JavaScript. SPLASH-I In Systems, Programming, Languages and Applications: Software for Humanity, 2015.

Download https://github.com/joaocampinhos/niverso

Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR
Trabalho apoiado por NOVA LINCS ref. UID/CEC/04516/2013