# Resolução distribuída de interacções de serviços na Internet

**Fernando Miguel Carvalho**
ISEL, Dept. de Engenharia Electrónica e Telecomunicações e de Computadores
R. Conselheiro Emídio Navarro, 1, 1949-014 Lisboa
mcarvalho@cc.isel.ipl.pt

**Rui Gustavo Crespo**
IST, Dept. of Electrical Engineering and Computers
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
R.G.Crespo@comp.ist.utl.pt

***Resumo***
*A proliferação da Internet levou ao crescimento substancial do número de serviços disponibilizados em aplicações como Email, VoIP e WWW, que conduziu à inevitável ocorrência de interacções, com comportamentos indesejáveis.*
*A resolução das interacções de serviços na Internet deve obedecer às características deste meio e como tal deve ter uma arquitectura distribuída. O projecto desenvolvido nesta dissertação apresenta uma solução baseada numa arquitectura distribuída, onde cada nó recorre à resolução prestada por um conselheiro[1].*
*Finalmente é apresentada uma proposta para a implementação do conselheiro com utilização de fórmulas deônticas e assente em tecnologia Java. O funcionamento da solução foi testado e adaptado com a aplicação de correio electrónico JAMES (Java Apache Mail Enterprise Server) da Apache Software Foundation (ASF).*
*Os resultados obtidos demonstraram o funcionamento da solução de acordo com os objectivos do projecto e o desempenho do sistema não comprometeu o nível de serviço da aplicação de Email.*

## Palavras Chave
Interacções de serviços, resolução distribuída, fórmulas deônticas, orientado por objectos, Java, JAMES.

## Introdução
Por variados motivos, desde comerciais, até à facilidade de criação de novos serviços e grande variação nos interesses dos utilizadores, as aplicações Internet existentes vão sendo complementadas cada vez mais com novos serviços. O problema surge quando um novo serviço interfere com um outro já existente, originando um resultado indesejado. A combinação de serviços, que resulta num comportamento inesperado, foi identificada pela primeira vez nos sistemas telefónicos e é designada por interacção de serviços ("feature interaction" [2]).

O problema das interacções de serviços tem sido estudado em vários tipos de aplicações Internet como Email [3], VoIP [4] e WWW [5].

Como exemplo de um caso de interacção de serviços, considere-se a aplicação de correio electrónico e três utilizadores Ana, Beatriz e Carlos, conforme apresentado na Figura 1. Supondo que a Ana subscreve o serviço de bloqueio de mensagens de e para o Carlos ("FilterMessage") e a Beatriz subscreve o serviço que redirecciona todas as mensagens recebidas ("ForwardMessage") para o Carlos, então se a Ana enviar uma mensagem para a Beatriz esta será reencaminhada e recebida pelo Carlos, violando o serviço subscrito pela Ana.

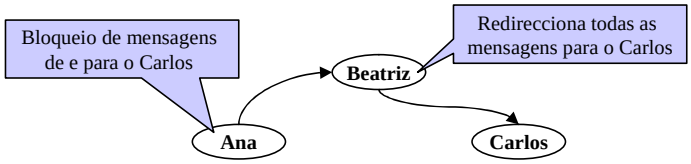


**Figura 1. Exemplo de interacção de serviços numa aplicação de *Email*.**

A interacção de serviços envolve três problemas distintos: **impedimento**, **detecção** e **resolução**. O **impedimento** implica que as aplicações sejam concebidas de forma a que seja prevista a interacção entre serviços antes da sua execução. A **detecção** consiste num processo de identificação de interacções através de métodos apropriados. A **resolução** é feita sobre situações de interacção de serviços previamente identificadas.

As características de distribuição e heterogeneidade da Internet, tornam as soluções de impedimento e detecção num nível de complexidade impraticável. Assim, estas soluções são confinadas a ambientes específicos, a um tipo limitado de aplicações e protocolos.

Trabalhos recentes na área da resolução distribuída, adoptam uma abordagem de dois níveis de descrição dos serviços, funcionamento e resolução [6]. No entanto esta solução exige a reprogramação aos dois níveis, para resolução de novas situações de interacção.
Uma outra implementação designada FIMA ("Feature Interaction Management Agents") [7], usa agentes para coordenação das operações de resolução. Contudo a falha de um FIMA, compromete toda a resolução.

Neste artigo, propomos uma solução onde cada instância de uma aplicação Internet, em cada nó, recorre à resolução prestada por um **conselheiro** [1]. O conselheiro é a entidade que tem conhecimento das situações de interacção de serviços, que são descritas por fórmulas lógicas. A solução foi implementada em tecnologia Java e o funcionamento testado e adaptado com a aplicação de correio electrónico JAMES (*Java Apache Mail Enterprise Server (a.k.a. Apache James)*) da *Apache Software Foundation* (ASF).

## Conselheiro

Neste artigo é identificada a arquitectura e a implementação da resolução distribuída de interacções de serviços na Internet, doravante identificado apenas por **sistema de resolução**. No modelo proposto cada instância de uma **aplicação** Internet pertence a um nó com um **conselheiro**, que serve todas as instâncias desse nó. Por sua vez, os conselheiros podem comunicar entre si para resolver interacções de serviços entre instâncias de aplicações de diferentes nós. Neste modelo, ao contrário da solução centralizada, a paragem de um conselheiro apenas afectará as instâncias do nó que este serve e as mensagens processadas pelo nó atingido pela falha.

Ao contrário do modelo centralizado, o conselheiro proposto tem características de escalabilidade, que permitem a introdução de novos conselheiros no processo de resolução de interacções sem que sejam necessárias alterações de programação. Esta, além de ser uma solução escalável é independente da implementação dos serviços e flexível na integração com outros tipos de conselheiros, desde que assentes no mesmo protocolo de comunicação.

O papel do conselheiro em cada nó é indicar às aplicações que serviço deve ser executado, entre os serviços candidatos. O processo é desencadeado com a aplicação que envia ao conselheiro uma lista de serviços candidatos a serem executados.

O conselheiro determina qual o serviço ou serviços a ser executados, com base num conjunto de **fórmulas** lógicas e através de métodos de resolução **deôntica**. O conjunto de fórmulas pode ser carregado dinamicamente. Ou seja, o conselheiro não tem um comportamento estático dependente da forma como foi programado. O seu comportamento é determinado pelas fórmulas que são introduzidas pelo seu administrador.

Nas duas subsecções seguintes será descrito a forma de representação dos serviços e das fórmulas deônticas.

### Representação dos Serviços

A variedade de serviços existente é vasta e está sujeita a novas actualizações. Como tal, no sistema de resolução um serviço é definido pelo conjunto de **acções** que são executadas por esse serviço. Assim, um serviço poderá ser representado por apenas uma, ou várias acções base. Esta abordagem permite, que nalgumas situações um novo serviço possa ser definido a partir das acções base já definidas, sem necessidade de actualização do conjunto de acções.

No modelo proposto para o sistema de resolução, os serviços são representados numa linguagem de predicado de primeira ordem [8], da seguinte forma: Um serviço *Fi* descrito por duas acções *A* e *B* é representado por *Fi(A, B)*, ou por *Fi(B, A)*. Nesta representação as acções *A* e *B* são constantes.

O identificador de uma acção começa por uma letra maiúscula e pode ter um parâmetro *_init*, referente ao nó emissor da correspondência, ou *_dest*, do nó destinatário. Por exemplo, o serviço de resposta automática a um *Email* ("AutoResponder") é representado pelas acções de *Delivery(_init)* e *Send(_init)*.

**Fórmulas Deônticas**

Quando uma aplicação envia ao conselheiro uma lista de serviços candidatos a serem executados, o conselheiro baseia-se num conjunto de **fórmulas deônticas** para determinar qual ou quais os serviços que poderão ser executados (fórmula 2.1).

**(2.1)** ***Acções Ù Condições à Restrição***

As fórmulas são definidas pela conjunção de **acções** e **condições** que resultam numa **restrição** (interdição de uma acção). A interdição de uma acção é formulada pelo operador deôntico *I*. Quando uma fórmula é satisfeita (i.e, o valor é `true`), então haverá uma acção que é interditada, com consequente interdição dos serviços cuja representação inclui a acção interditada.

Uma fórmula é satisfeita se os seus serviços especificados fizerem parte da lista de serviços enviados pela a aplicação e ainda se as condições que fazem parte dessa fórmula também forem satisfeitas. Uma vez que os serviços são definidas por acções, significa que se os serviços de uma fórmula são iguais aos serviços candidatos de uma aplicação, então as suas acções são as mesmas. Logo o processo de emparelhamento entre os serviços candidatos e as fórmulas, pode ser aplicado ao nível da acção. Por esta razão, os serviços aparecem representados nas fórmulas pelo conjunto de acções que os definem.

Uma condição tem a mesma representação de uma acção mas a primeira letra do predicado é uma letra minúscula. Tal como as acções, podem receber um parâmetro *_init* ou *_dest*, que representa o nó emissor ou destinatário da correspondência, respectivamente. Por exemplo, uma condição que traduz o estado de uma aplicação *VoIP* que tem uma chamada em espera é representada por: *one_hold(_init).*

As condições dividem-se em três tipos de restrições:

- **Restrição Local** - O conselheiro verifica a condição com base na informação do seu nó, como é o caso do estado de uma aplicação desse nó.
- **Restrição em Cadeia** - Uma correspondência que é reencaminhada automaticamente entre vários utilizadores, deverá acumular os endereços por onde passou. A detecção de um endereço em duplicado na cadeia de endereços, indicia a detecção de um ciclo infinito.
- **Restrição Ponto a Ponto** - A verificação de uma condição está dependente de um conselheiro de outro nó.

Como exemplo de uma restrição local, considere-se a situação de um utilizador que tem subscrito o serviço de bloqueio de mensagens ("FilterMessage") e de reencaminhamento todas as mensagens (“ForwardMessage”). Numa situação de de bloqueio de uma mensagem, não faz sentido que essa mensagem seja reencaminhada por esse utilizador. A fórmula de restrição local 2.2, interdita o reencaminhamento, em caso de filtragem da mensagem. Nesta fórmula o serviço *FilterMessage* é representado pela acção *Deny*.

**(2.2)** ***Forward(_init, _dest) Ù Deny(_init) à I Forward(_init, _dest)***

No caso da detecção do ciclo infinito, provocado pelo serviço de reencaminhamento de mensagens entre dois utilizadores, a fórmula de restrição em cadeia 2.3, impede e execução da acção de *Forward*.

**(2.3)** ***Forward(_init, _dest) Ù loop(_init, _dest) à I Forward(_dest)***

Na resolução da situação apresentada na Figura 1, o emissor original adiciona uma directiva à mensagem, que será confirmada no destinatário, pela condição *permission(_init)* da fórmula de restrição ponto a ponto 2.4. No exemplo da Figura 1, o reencaminhamento seria interditado porque o emissor não daria a permissão necessária, satisfazendo a condição ¬ *permission(_init)* (¬ é o operador de negação).

**(2.4)** ***Forward(_init, _dest) Ù ¬permission(_init) à I Forward(_dest)***

# Arquitectura

No processo de resolução de interacções de serviços, o conselheiro interage com aplicações Internet e ainda com outros conselheiros, da seguinte forma:

1. Recebe das aplicações Internet listas de **serviços candidatos** a serem executados;
2. Envia **conselhos** às aplicações Internet desse nó, sobre a execução, ou não, de determinados serviços em interacção;
3. Pede a outros conselheiros de outros nós informação necessária à avaliação de determinadas **directivas**.

4. Envia um **pedido** a uma aplicação para avaliação de uma directiva, que foi remetida por um conselheiro de outro nó, acerca de uma mensagem enviada a partir deste nó.
5. Pede a uma aplicação informação complementar, para avaliação de uma determinada condição (Ex: listas de endereços a barrar);

A resposta a uma aplicação sobre a execução de um determinado serviço é feita de modo assíncrono. Isto é, uma aplicação Internet pode enviar diversos pedidos de resolução de interacção de serviços que serão respondidos pelo conselheiro independentemente da sua ordem de chegada. Cada pedido de resolução desencadeia um fio de execução separado ("thread"), que terminará com a resposta do conselheiro à aplicação.

As interacções com origem num conselheiro, para outro conselheiro distinto ou uma dada aplicação (3º, 4º e 5º ponto, respectivamente), são desencadeadas na sequência de um pedido de resolução feito por uma aplicação. Por isso estes pedidos são síncronos e terminam apenas com a obtenção de uma resposta, ou a conclusão de um período de tempo limite.

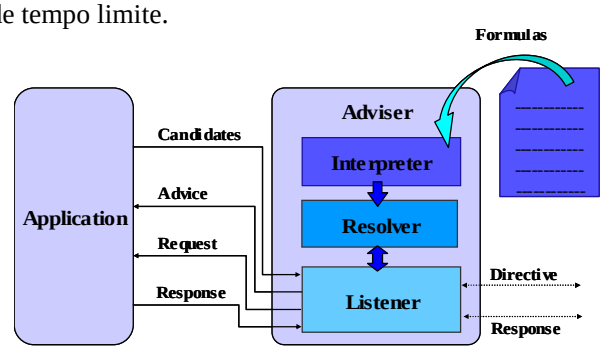


**Figura 2. Diagrama de componentes do conselheiro**

Para responder aos requisitos apresentados, e para modelar o sistema em unidades distintas de modo a facilitar a sua compreensão e desenvolvimento, o conselheiro tem uma arquitectura assente em três componentes apresentados na Figura 2, com a seguinte designação:

- ***Interpreter*** - Este componente actua sobre o conselheiro na altura do seu arranque ou sempre que houver uma alteração do ficheiro de entrada, que contém as **fórmulas** deônticas. Estas fórmulas intervêm no comportamento do **decisor**.
- ***Resolver*** - Módulo de resolução das interacções de serviços, responsável pelo processamento da lista de **candidatos** enviada pelas aplicações Internet e fazer a avaliação dos serviços em interacção, face às **fórmulas** carregadas. Neste processo de avaliação o decisor poderá ter que fazer **pedidos** a outros conselheiros ou à aplicação, para validar uma determinada **directiva** ou **condição** respectivamente.
- ***Listener*** - Responsável por todo o processo de comunicação com as aplicações Internet e outros conselheiros, desde o estabelecimento dos canais de ligação, escuta, envio e processamento das mensagens XML em instâncias de classes Java. Neste processo decorrem diversos fluxos de troca de mensagens.

### Interpreter

O interpretador é responsável pelo processamento de um conjunto de fórmulas deônticas descritas numa linguagem de programação, especificada pela sintaxe EBNF [9], seguinte:

*formula_list* **à** *formula*$^{+}$
*formula* **à** *left "=>" status ";"*
*left* **à** *actions | conditions | actions "&&" conditions*
*actions* **à** *ACTIONS id_list*
*conditions* **à** *CONDITIONS condition*$^{+}$
*condition* **à** *"(" ID id_list ")"*
*status* **à** *order id_list*
*order* **à** *INTERDICTIONS | PERMISSIONS*
*id_list* **à** *"(" ID*$^{+}$ *")"*

Por exemplo, a fórmula 2.3 é descrita nesta linguagem pela fórmula 3.1.

**(3.1)** ***actions(Forward) && conditions(conf.AdvLoop()) => interdictions(Forward);***

As acções são definidas por *strings* constantes e as condições pelos identificadores das classes, que derivam de `AbstractCondition`, e cujas instâncias podem ser inicializados com os argumentos passados entre parêntesis, como é o caso da fórmula 3.2, que interdita a filtragem de uma mensagem caso tenha origem governamental. Ou ainda, a fórmula 3.3, que interdita o serviço de resposta automática (representado pela combinação das acções *Delivery* e *Send*), quando o emissor é o endereço de uma lista de distribuição.

**(3.2)** ***actions(Deny) && conditions(conf.FromHost(gov)) => interdictions(Deny);***
**(3.3)** ***actions(Delivery, Send) && conditions(conf.FromHost(yahoogroups.com)) => interdictions(Send);***

**Resolver**

A determinação do conselho é feita com base num conjunto de **fórmulas** lógicas, instâncias do tipo `irs.adv.rsv.Formula`. Por sua vez, a lista de serviços candidatos é recebida numa **mensagem**, que é uma instância do tipo `irs.msg.jaxb.MsgCandidates`. A validação de cada fórmula sobre a lista de candidatos é dada pelo método `validate` de `irs.adv.rsv.Formula`, do qual resulta um conjunto de acções interditadas. Da união dos vários conjuntos de acções resultantes da validação de cada uma das fórmulas, obtém-se o conjunto final de acções interditadas, que levará consequentemente à determinação dos serviços interditados.

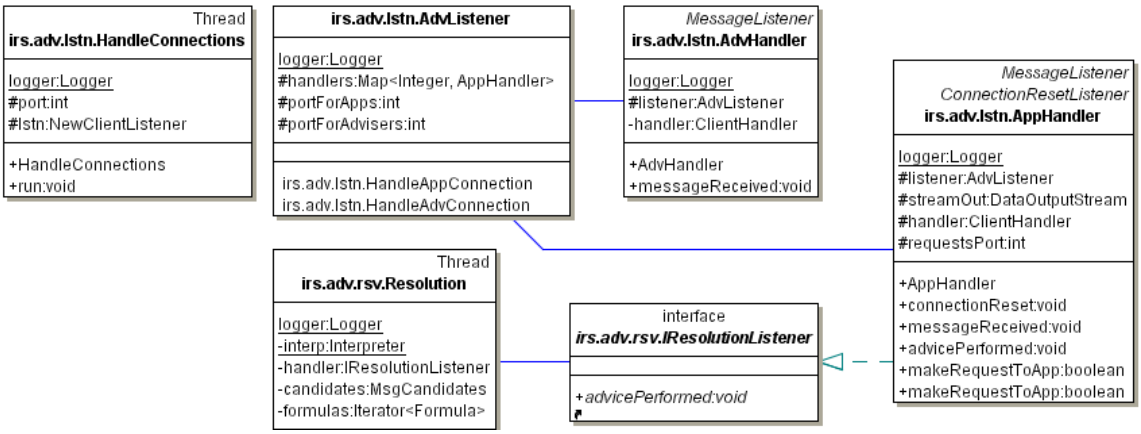


**Figura 3. Diagrama UML de classes dos componentes *Resolver* e *Listener***

Por cada lista de candidatos enviada por uma aplicação é lançada uma nova *thread* correspondente a uma nova **resolução**, que termina com a determinação de um **conselho**. Cada resolução é definida por uma instância da classe `irs.adv.rsv.Resolution`, que guarda a informação relativa à lista de serviços candidatos e às fórmulas a validar, nos atributos `candidates` e `formulas` respectivamente, conforme apresentado na Figura 3. Uma vez processadas todas as fórmulas é apurado o conjunto de serviços em interacção, a retornar à aplicação. No final é notificado o transmissor, através da referência para irs.adv.rsv.IResolutionListener.

**Listener**

A instância da classe `irs.adv.lstn.AdvListener` desencadeia o processo de estabelecimento das ligações, recepção e envio de mensagens com as aplicações Internet e outros conselheiros. São lançadas duas *threads*, uma para tratamento das ligações das aplicações Internet e outra das ligações de outros conselheiros. Cada uma destas *threads* é representada por uma instância da classe `irs.adv.lstn.HandleConnections`(Figura 3).

Após o estabelecimento de uma ligação a uma aplicação Internet ou outros conselheiro, é instanciado um objecto de `AppHandler` ou `AdvListener` (Figura 3), respectivamente, que será o *handler* para os serviços candidatos recebidos (notar que esta é a classe que implementa `MessageListener` e cuja a instância será registada como *listener* das mensagens recebidas neste canal). Por cada aplicação Internet ligada existe uma instância de `AppHandler` correspondente, controlado por `AdvListener`.

## Resultados e Conclusões

O sistema de resolução foi integrado e testado com o servidor de Email James, tendo respondido a todos os requisitos propostos e apresentados neste artigo. Além disso é importante avaliar o custo de processamento das

mensagens de Email, com a intervenção do conselheiro. Foram testadas e avaliadas diferentes situações, sendo aqui apresentados os valores médios registados para quatro casos:

1. O destinatário tem subscrito os serviços de *FilterMessage, ForwardMessage* e *AutoResponder*. Quando uma mensagem é filtrada, a fórmula 2.2 e outra similar, interdictam os serviços de *ForwardMessage* e *AutoResponder*.
2. Referente à fórmula 3.2, que interditará o serviço *FilterMessage*;
3. Referente à fórmula 3.3, que interditará o serviço *AutoResponder*;
4. Exemplo da Figura 1 e resolvido pela fórmula 2.4.

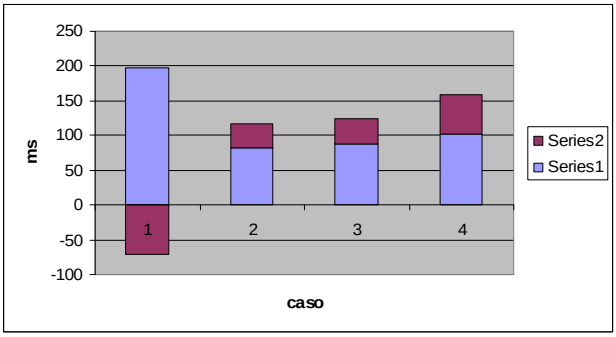


**Figura 4. Variação do tempo médio de processamento de *Email* com intervenção do conselheiro.**

Conforme consta no gráfico da Figura 4, no primeiro caso há uma diminuição no tempo de processamento das mensagens, devido a uma redução do número de serviços executados. Nos outros casos, o aumento no tempo médio de processamento da mensagem varia entre os 42 e 55%. Este acréscimo de tempo é devido, na sua maioria, à comunicação e serialização das mensagens. Além disso, este custo é fixo sendo no máximo de 50 ms e representa a quase totalidade do tempo de resolução.

A solução foi desenvolvida sobre a plataforma J2SE versão 5. Foram usadas ainda as ferramentas *JLex* e *Cup* [9], para construção de analisadores léxicos e sintácticos do *Interpreter* e a plataforma JAXB, ("Java Architecture for XML Binding") [10], para a definição dos tipos e processo de serialização das mensagens XML transmitidas entre os conselheiros e as aplicações Internet.

Em suma a solução responde aos requisitos propostos, sem comprometer o nível de serviço da aplicação Internet. Apresenta uma arquitectura distribuída e a potencialidade da resolução não ser comprometida por nós sem conselheiros, potenciando a perspectiva de introdução faseada dos conselheiros na Internet

## Referências


[1] Carvalho, F. M e Crespo, R. G. "An experimental distributed resolution of WWW interactions", *Proceedings of IADIS International Conference WWW/Internet 2005*, Lisbon, Portugal.

[2] Bowen, T.F., et al., 1989. "The Feature Interaction Problem in Telecommunication Systems", *Proceedings of 7th Int'l Conference on Software Engineering for Telecommunication System*s, pp 59-62.

[3] Hall, R.J., 2000, "Feature Interactions in Electronic Mail", *Proceedings of 6th Int'l Workshop on Feature Interations in Telecommunication and Software Systems*, Glasgow, Scotland, pp 67-82.

[4] Lennox, J. and Schulzrinne, H., 2000, "Feature Interaction in Internet Telephony", *Proceedings of 6th Int'l Workshop on Feature Interactions in Telecommunication and Software Systems*, Glasgow, Scotland, pp 38-50.

[5] Weiss, M., 2003, "Feature Interactions in Web Services", *Proceedings of 7th Int'l Workshop on Feature Interations in Telecommunication and Software Systems*, Ottawa, Canada, pp 149-156.

[6] Pang, J. and Blair, L., 2003, "Separating Concerns from Distributed Feature Components", *Proceedings of Workshop on Software Composition*, Warsaw, Poland

[7] Chentouf, A. et al, 2003. "Experimenting with Feature Interaction Management in SIP Environment", *Telecommunication Systems*, Vol. 24 No. 2, pp 251-274.

[8] Hamilton, A. G., *Logic for Mathematician*, Cambridge University Press, 1988.

[9] Appel, A. W., 1998, *Modern Compiler Implementation in Java*. Cambridge University Press. UK.

[10] McLaughlin, B., 2002, *Java & XML Data Binding*, O'Reilly. Sebastopol, USA.