Reconfiguração Dinâmica de Modelos de Interação para Redes de Sensores

Adérito Baptista, Maria Cecília Gomes e Hervé Paulino

CITI / Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Resumo As Redes de Sensores Sem Fio (RSSF) são hoje em dia uma tecnologia fundamental para a ciência em todos os domínios, permitindo um sensoriamento em larga escala do ambiente e a um custo baixo. Redes formadas a partir de sensores fixos ou móveis, permitem fornecer dados com maior precisão a aplicações científicas já existentes e, potenciam o desenvolvimento de novos tipos de aplicações, como sejam a utilização de dispositivos móveis para a recolha e disseminação de informação para aplicações em Sensoriamento Participativo. Tendo em vista a integração das RSSF em ambientes Web, foi realizada a sua abstração como serviços Web[1,2]. No entanto, são ainda limitados tanto os modelos de interação oferecidos por essas *interfaces* bem como os mecanismos disponíveis para a sua adaptação em tempo de execução. Neste contexto, é discutida a relevância da utilização de Padrões de Desenho no desenho de um *middleware* que permite também que essas novas formas de interação possam variar dinamicamente e de acordo com o contexto subjacente aos utilizadores do serviço. É descrita ainda uma avaliação funcional do mesmo, através do desenvolvimento de uma aplicação que faz uso dos mecanismos de reconfiguração dinâmica implementados no *middleware*.

Keywords: Redes de Sensores Sem Fio, Serviços *Web* com estado, Sistemas auto-adaptáveis, Padrões de desenho

1 Introdução

Nunca, como agora, foi tão necessária a monitorização do ambiente de modo a fornecer, a diferentes tipos de aplicações, dados com maior precisão e, de preferência, atualizados em tempo-real. Por exemplo, o denominado "efeito de estufa"tem gerado condições meteorológicas extremas na forma de inundações em larga escala, tornados, etc, tornando-se crucial que, aplicações que simulam eventos neste contexto, a) tenham acesso a diferentes tipos de dados, b) os dados sejam recolhidos em períodos alargados no tempo, e c) em zonas geográficas de larga escala. Para este efeito, as Redes de Sensores Sem Fio (RSSF), consistindo num elevado número de sensores (mais ou menos simples) espalhados

pela zona geográfica a avaliar, oferecem uma boa solução de baixo custo para a monitorização do ambiente em larga escala.

As RSSF, formadas a partir de sensores fixos ou móveis, são hoje uma tecnologia fundamental para a ciência em todos os domínios, quer dando suporte a aplicações científicas mais tradicionais, quer potenciando o desenvolvimento de novos tipos de aplicações. Um exemplo deste segundo caso, é a utilização de dispositivos móveis para a recolha e disseminação de informação para aplicações em *Sensoriamento Participativo* [3] (e.g. aplicações de gestão de tráfego automóvel).

Porém, um dos problemas relacionados com as RSSF é a sua interface de baixo nível. Assim, têm sido propostas abstrações de nível mais alto que permitem, por exemplo, ver a rede como uma base de dados, um fluxo de dados ou um conjunto de agentes. Outra solução é a abstração das redes como serviços *Web* [1,2], o que permite que as RSSF sejam integradas em ambientes *Web*, e.g. em processos de negócio. Deste modo, o paradigma dos serviços permite o seu acesso uniforme e simplificado através de tecnologias *Web standard*, sistematizando o modo de acesso aos dados por elas recolhidos, ou ainda a sua própria parametrização e agregação.

A importância do paradigma dos serviços '*per si*' tem aumentado em várias áreas, existindo mesmo uma tendência de disponibilizar "tudo como um serviço" (*XaaS - everything as a Service*). Áreas recentes têm também contribuído para esta crescente relevância, como por exemplo a Computação Ubíqua, ou a *Internet das Coisas* [4]. Um serviço é hoje visto como uma abstração simples, mas poderosa, para a tão necessária interação entre sistemas, disponibilizando uma forma uniforme de acesso a, e associação de, entidades com características bem distintas, e a diferentes níveis da *ciber-infraestrutura*.

No entanto, verifica-se a necessidade de novas soluções e modelos para a interação com serviços, incluindo os que representem RSSF. Estes tipicamente providenciam o acesso a recursos com estado, requerendo por isso interações mais complexas que o modelo tradicional de interação pedido/resposta. Adicionalmente, é também crescente a necessidade da existência de mecanismos dinâmicos de adaptação em termos da interação com serviços, quer como forma de garantir tolerância às falhas, quer por necessidade dos seus utilizadores controlarem tais interações.

Dimensões do Problema

Uma das características do modelo de serviços, que muito contribuiu para a sua simplicidade e larga aceitação, foi o modo usual de interação pedido/resposta sem estado, presente, por exemplo, nos serviços *Web* na sua forma primitiva. No entanto, tem sido crescente a utilização de serviços no acesso a recursos com estado, e.g. para obtenção de informação de aplicações em execução ou sobre o estado dos recursos. Este foi o caso dos serviços de computação em *Grid*, onde as aplicações são, tipicamente, de longa duração [5]. Tal deu origem à norma *Web Services Resource Framework (WSRF)* [6], que define como aceder a atividades continuadas através de serviços *Web*, sendo tais acessos também necessários no contexto de computação em *Cloud* (e.g. avaliação dos recursos consumidos pela

aplicação ao qual estará associado um custo acumulado). Este tipo de serviços de acesso a recursos com estado, requer que a sua implementação tenha em consideração a existência de um estado dinâmico (i.e. que varia) [5]. É assim da responsabilidade do serviço, a manutenção deste estado ao longo do tempo (i.e. entre várias trocas de mensagens com utilizadores do serviço).

Existe assim vantagem e necessidade na existência de modelos de interação mais complexos e flexíveis entre os prestadores de serviços e os seus utilizadores. Por exemplo, nos serviços *Web* para RSSF [2]: a) o modelo Editor/Assinante [7] permite que os utilizadores sejam notificados de eventos da RSSF que subscreveram (e.g. notificação que a temperatura média detetada por uma rede de sensores excedeu um limite pré-definido); b) o modelo Fluxo de Dados [7] permite a disseminação dos dados recolhidos pelos sensores da rede.

Porém, são ainda limitados tanto os modelos de interação oferecidos por essas interfaces, bem como os mecanismos disponíveis para a sua adaptação em tempo de execução. Um exemplo é a necessidade que os utilizadores poderão ter de, dinamicamente, agregar diferentes Fluxos de Dados (e.g. à semelhança dos *mashups*) gerados por redes RSSF distintas mas relacionadas. Por exemplo, RSSF que recolhem dados sobre a temperatura na mesma zona geográfica mas a diferentes alturas do solo, e ainda de RSSF que recolhem dados sobre os níveis de humidade nessa mesma zona. Assim, em vez da aplicação cliente ter de aceder explicitamente a serviços *Web* distintos, há vantagem em que esse acesso se faça de uma forma mais simples permitindo a agregação da informação.

Há assim benefícios em que as aplicações baseadas em RSSF tenham um controlo mais rico e flexível em termos da interface com o serviço. Para tal, devem ser considerados: a) *o contexto dessa interação*, o que inclui o contexto do utilizador do serviço (e.g. um dispositivo móvel com pouca bateria ou que muda de zona geográfica), o contexto da rede de comunicação entre o serviço e o seu utilizador (e.g. as características da rede de comunicação), ou o contexto do próprio serviço (e.g. estado da aplicação em execução, ou disponibilidade dos recursos da arquitetura de suporte à sua execução), e; b) *a possibilidade dessa interação evoluir no tempo*, com reconfiguração despoletada automaticamente pelo sistema de suporte ao detetar que o contexto variou (e de acordo com regras bem definidas), ou então por pedido explícito do utilizador do serviço.

De modo a lidar com estes problemas, as próximas secções descrevem uma solução baseada nos conceitos de Padrões de Desenho e de Arquitetura [8,7], e a sua implementação, sendo usado o contexto dos serviços *Web* para RSSF como domínio de aplicação. Finalmente, são apresentadas as conclusões e trabalho futuro.

2 Reconfiguração Dinâmica baseada em Padrões

O conceito de *padrão* é considerado nas dimensões de *estrutura* e *comportamento*, na forma de esqueletos (ou *templates*) parametrizáveis, tal como descrito em [9], permitindo flexibilidade tanto na sua composição como na sua reconfiguração dinâmica. Os esqueletos de Padrões de Estrutura definem a semântica

das ligações estruturais entre os elementos que deles fazem parte, sem no entanto especificarem quaisquer restrições em termos de fluxo de dados ou de controlo entre esses elementos. Esses padrões incluem agregações tais como topologias (e.g. *pipeline*, estrela, anel) bem como *Padrões de Desenho* [8] (e.g. *Façade*, *Proxy*, etc.). A reconfiguração estrutural permitida para cada um destes padrões é específica e restrita à sua definição, garantindo que os padrões mantêm a sua semântica. Por exemplo, é possível alterar o número de sub-sistemas que constituem um *Façade*, mas o padrão reconfigurado tem de continuar a possuir um elemento central responsável pela a interface com os seus subsistemas.

Já os esqueletos de Padrões de Comportamento definem as dependências em termos de fluxo de dados e fluxo de controlo entre os elementos que constituem o padrão, bem como o papel desses elementos no contexto do padrão (e.g. no caso do padrão Editor/Assinante existe a definição dos papéis de editor e assinante, bem como dos fluxos de dados e controlo sob a forma de notificações assíncronas nos assinantes). Outros padrões utilizados são o Fluxo Contínuo de Dados, Produtor/Consumidor, e Cliente/Servidor. Novamente, todos estes padrões são reconfiguráveis, e as reconfigurações possíveis são tais que garantem a consistência do padrão (e.g. no contexto do padrão Produtor/Consumidor, pode alterar-se o número de produtores e de consumidores, mas tem de continuar a existir pelo menos uma entidade com papel de produtor, e idem para o papel de consumidor). Adicionalmente, a separação entre estrutura e comportamento, permite a combinação de um padrão de estrutura com diferentes padrões de comportamento (e vice-versa), bem como a substituição dinâmica de um padrão de comportamento por outro [9].

Finalmente, o modo como é definido o contexto da interação, e sua reconfiguração, com um serviço de acesso a recursos com estado, é através da noção de Sessão. Uma Sessão inclui: a) a composição de um padrão de estrutura (geralmente um *Façade*) com um padrão de comportamento particular, o que representa o modelo de interação a ser garantido para todos os utilizadores desse serviço e, no contexto dessa Sessão particular (e.g. Agregação de eventos gerados por diferentes redes de sensores em determinados tópicos, e a sua entrega aos utilizadores constantes da Sessão, de acordo com o modelo Editor/Assinante); b) um identificador que permite, por exemplo, que novos utilizadores se possam juntar a essa Sessão; c) os identificadores dos utilizadores que, num dado momento, pertencem à Sessão; d) o identificador do "dono da Sessão"(o seu utilizador mais antigo, e que pode, dinamicamente, requerer uma reconfiguração explícita do modelo de interação a utilizar nessa sessão); e) quais os mecanismos possíveis de adaptação dinâmica estruturada e dependente do contexto, o que engloba o serviço, os seus utilizadores, e a sua interação (e.g. é possível definir qual o novo padrão de comportamento a utilizar automaticamente caso existam falhas ou alterações na qualidade da comunicação com os membros da Sessão); f) o tempo de vida de uma sessão, findo o qual uma sessão deixa de existir. Esta noção de Sessão e possíveis reconfigurações serão ilustradas nas próximas secções, bem como a sua implementação.

2.1 Exemplo de Aplicação

Consideremos o seguinte cenário: várias RSSF encontram-se distribuídas numa floresta capturando informação de diversos tipos - vento, temperatura, humidade, etc, a qual é acessível através de um serviço com estado. Ao invés de periodicamente consultar a informação disponível de acordo com o modelo de interação Cliente/Servidor, a corporação de bombeiros responsável por essa floresta cria uma Sessão com um modelo de interação Editor/Assinante, tal que exista uma notificação caso a temperatura aumente para valores próximos de incêndio. Para mais, indica qual o padrão de reconfiguração automática quando esse aumento se verifica – Agregação de Fluxo Contínuo de Dados dos valores de temperatura, e ainda direção e velocidade do vento. Em caso de incêndio, a notificação é enviada para a corporação de bombeiros, e a Sessão é automaticamente reconfigurada para a agregação de Fluxos Contínuos de Dados de temperatura, bem como de direção e velocidade do vento, recolhidos na zona. É assim possível obter o estado do incêndio e a sua direção de propagação. Adicionalmente, novas corporações de bombeiros destacadas para ajudar no combate ao incêndio, poderão juntar-se a essa Sessão, recebendo assim a mesma informação atualizada, o que facilitará a sua inter-coordenação. É também possível que a corporação de bombeiros dona da Sessão, requeira um Fluxo Contínuo de Dados adicional sobre a humidade da zona, de modo a possuir um maior grau de certeza sobre a evolução do sinistro. Automaticamente, todos as outras corporações constantes da Sessão passarão a ter também acesso a esses dados.

3 *Middleware* para Acesso Adaptável a RSSF

Esta secção introduz um *middleware* para o acesso adaptável a RSFF, que oferece, por um lado, um conjunto de modelos de interação estruturados na forma de Padrões e suportados pelo conceito de Sessão, e por outro, mecanismos de reconfiguração dinâmica estruturada (também baseada em padrões) desses modelos. A reconfiguração pode incidir no contexto do modelo de interação corrente, bem como na substituição de um modelo por outro.

A arquitetura do *middleware* segue um modelo *three-tier* (Figura 1) em que a apresentação, a lógica e as fontes de dados estão desacopladas. Faz-se de seguida uma descrição geral desta arquitetura, para enquadramento da descrição posterior, mais detalhada, dos componentes em cada nível.

No que se refere à *camada de apresentação*, a interface do *middleware* está acessível via serviços *Web*, tendo, no entanto, sido desenvolvida uma API para aplicações cliente Java que esconde os detalhes da comunicação *Web*. A descrição detalhada desta API do utilizador para os serviços disponibilizados, encontra-se na Subsecção 3.3.

A *camada lógica* fornece os modelos de interação e a sua reconfiguração dinâmica. Os modelos de interação que consideramos relevantes no contexto das RSSF são os Padrões já referidos – Cliente/Servidor, Editor/Assinante, Produtor/Consumidor, Fluxo Contínuo de Dados, e, ainda, Agregação dos anteriores.

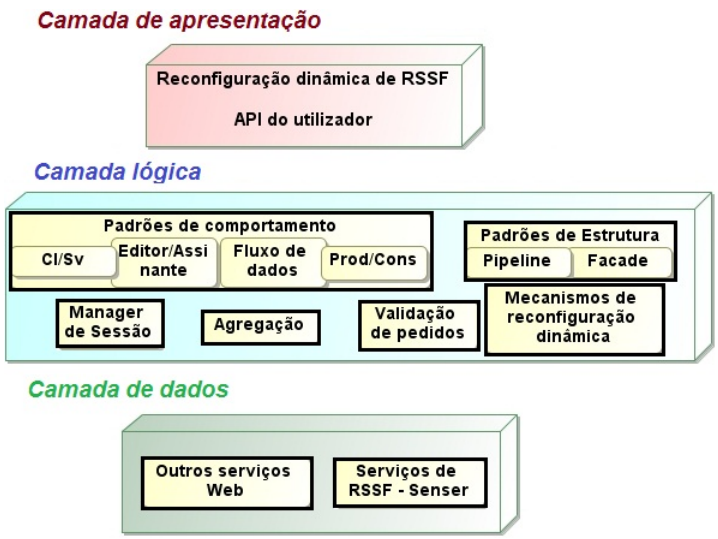


Figura 1: Arquitectura de três camadas

A sua escolha prende-se com o facto de representarem modelos de fluxo de dados com "qualidades de serviço"distintas, permitindo bastante diversidade no acesso aos dados gerados pelas redes.

A *camada de dados* acede a outros sistemas via serviços *Web* e fornece-os à camada lógica. Como apresentado na Figura 1, este acesso às fontes de dados aplicacionais via serviços *Web* inclui os serviços disponibilizados pela plataforma *SenSer* [2]. Esta é a plataforma que é utilizada neste trabalho para integrar as RSSF no contexto *Web*, e que disponibiliza já modelos de interação Editor/Assinante e Fluxo Contínuo de Dados, para além do Cliente/Servidor, para acesso aos dados.

Quanto aos componentes da *camada lógica*, em particular, o papel de cada um é: *Padrões de Estrutura* – criação dos Padrões de Estrutura utilizados no contexto de uma sessão. *Padrões de Comportamento* – criação dos Padrões de Comportamentos usados no contexto de uma sessão, definindo o modelo de interacção entre um serviço e os seus utilizadores nesse contexto. *Agregação* – esta componente permite que se combinem múltiplas fontes de dados, e a forma como esta agregação é feita é parameterizável pelo utilizador; este pode também associar um comportamento ao resultado desta agregação. *Validação de pedidos* – esta componente assegura que os pedidos feitos pelos utilizadores são válidos. *Mecanismos de reconfiguração dinâmica* – estes mecanismos possibilitam transições automáticas bem definidas, entre estados, respeitantes a utilizadores contratados a uma sessão.

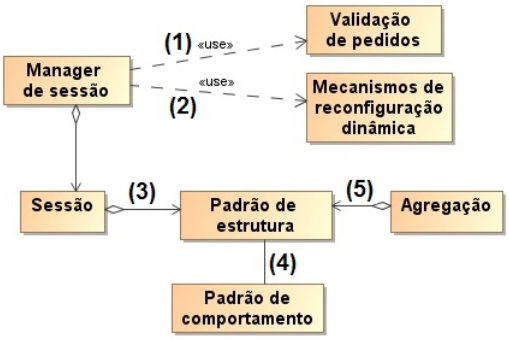


Figura 2: Relações entre os componentes

3.1 Implementação

A relação entre os componentes lógicos anteriormente descritos é ilustrada na Figura 2. O *Manager de Sessão* é responsável pela gestão das sessões existentes (*Sessão*) e fá-lo com recurso aos componentes de *Validação de pedidos* (1) e de *Mecanismos de reconfiguração dinâmica* (2). A validação de um pedido feito por um utilizador passa por efetuar as autenticações da sessão ao qual o pedido é endereçado, do utilizador e da operação no contexto da sessão. Os mecanismos de reconfiguração dinâmica são automaticamente despoletados perante situações onde é necessária a transição do estado corrente para outro. A subsecção 3.2 identifica essas mesmas situações e os respetivos mecanismos usados para a implementação de uma máquina de estados. Uma *Sessão*, por seu lado, é composta por, pelo menos, um *Padrão de estrutura* (3) e guarda dados relativos ao seu estado como previamente descrito (e.g. o dono da sessão, o(s) tópico(s) relacionado(s),etc.) Por sua vez, um padrão estrutural tem associado um comportamento (4).

Passamos a explicar em mais detalhe, os comportamentos por nós considerados: *Cliente/Servidor* – este é o modelo base de interação entre os utilizadores e o serviço. Através dele, os clientes podem realizar pedidos síncronos, como seja obter valores registados por sensores. *Editor/Assinante* – O cliente subscreve um tópico de interesse, e.g. *windspeed >40 km/h*, e é notificado quando tal evento acontece. Para além do tópico, o cliente pode indicar um padrão para o qual o modelo de interação é automaticamente reconfigurado, aquando da ocorrência do evento. *Fluxo Contínuo de Dados* – O cliente indica um tópico sobre o qual pretende obter um fluxo contínuo de dados. Estes dados são enviados a uma taxa definida pelo cliente. Este tipo de interação revela-se mais indicado para contextos onde é preferível *performance* em detrimento da qualidade. *Produtor/Consumidor* – Análogo ao Fluxo Contínuo de Dados com a diferença de existir memória tampão entre o utilizador e o serviço, permitindo alguns níveis de QoS associados a este padrão. A memória tampão existente entre ambos é independente do cliente e deteta quando o consumo não consegue acompanhar a produção de mesmos. Quando assim acontece, este pede a reconfiguração para

uma sessão com exatamente as mesma características, mas com menor taxa de envio. Adicionalmente, este modelo garante ordenação de mensagens.

Por último, o componente de Agregação combina vários padrões estruturais e comportamentais (5), todos inseridos no mesmo contexto de Sessão. Numa Agregação existem várias fontes de dados, e o comportamento associado a este *Facade* é específico, tendo a função de agregar a informação. Visto que pretendemos dar a liberdade ao utilizador de confinar os critérios de agregação (e.g calcular a média de valores, ou definir um *racio* pelas diferentes fontes), o cliente deverá fornecer uma função de agregação. Esta mesma função será aplicada aos dados provenientes das fontes. Deste modo, o seu resultado pode agora ser entregue a outro *Facade* responsável pela disseminação. E uma vez que existe um Fluxo Contínuo de Dados entre ambos os *Facades*, foi escolhido um *Pipeline* como estrutura de suporte para tal. Por fim, todo este processo insere-se no contexto de Sessão - Figura 3.

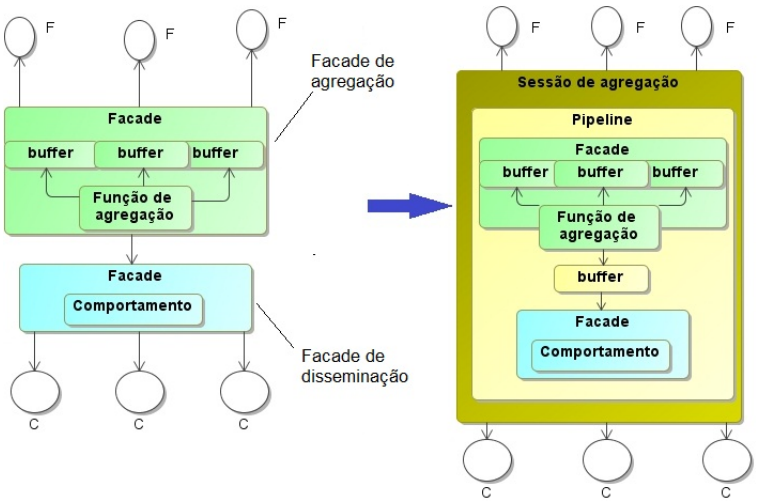
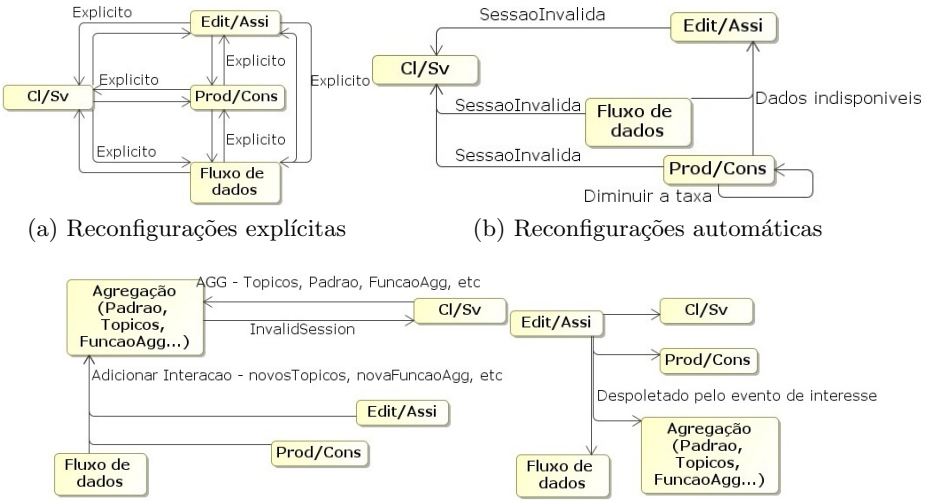


Figura 3: Sessão de agregação de dados

3.2 Mecanismos de Reconfiguração Dinâmica

A máquina de estados implementada pelo nosso *middleware* é ilustrada na figura 4. Ela foi subdividida em quatro como forma de facilitar a sua explicação. A *primeira máquina* (4a) está relacionada com *pedidos explícitos* feitos pelo utilizador, em termos da reconfiguração de um padrão para outro. Estas transições são diretamente implementadas pelo contexto de Sessão. Por exemplo, quando um utilizador particular pretende mudar o seu modelo de interação de acesso aos dados, e.g. de Editor/Assinante para Fluxo Contínuo de Dados, acontece o seguinte: ele é removido da primeira Sessão e se, já existir uma Sessão que



(a) Reconfigurações explícitas

(b) Reconfigurações automáticas

(c) Reconfigurações - Agregação

(d) Reconfigurações - Editor/Assinante

Figura 4: Máquina de estados

disponibilize o novo modelo de interacção pretendido para esses mesmos dados, o utilizador é adicionado a essa segunda Sessão. Senão, é criada uma nova Sessão com essas características (passando a ser este utilizador o dono desta nova Sessão). Sempre que uma Sessão deixa de possuir utilizadores, ela é destruída.

A *segunda máquina de estados* (4b) está relacionada com transições automáticas feitas pelo *middleware*, com o intuito de manter a consistência das interações, e por resposta a alterações no contexto do serviço, utilizadores, ou meio de comunicação entre eles. As transições ditas "*SessaoInvalida*"ocorrem quando: a) as sessões do tipo Fluxo Contínuo de Dados e Produtor/Consumidor expiram ou a fonte de dados dessa sessão não está disponível (*off-line*), ou; b) a subscrição do modelo Editor/Assinante torna-se inválida. As transições ditas "*Dados indisponíveis*"ocorrem quando o *middleware* deixa de receber dados a serem disseminados por uma Sessão, mas a fonte dos mesmo está "viva"(ao contrário das *SessaoInvalida*). Deste modo, os utilizadores são reconfigurados para *Editor/Assinante*, e assim que os dados voltem a ser enviados pela fonte, os utilizadores são notificados. A transição "*Diminuir a taxa*"ocorre quando a memória tampão do modelo Produtor/Consumidor, enquanto entidade independente do consumidor, verifica que o mesmo não consegue acompanhar a taxa com que os dados são enviados. O cliente é assim reconfigurado para uma Sessão com menor taxa. Esta transição será útil se imaginarmos que, no exemplo dos fogos, poderão existir clientes móveis (menor capacidade de processamento) que circulem pela floresta, e desejem obter dados sobre o perímetro do incêndio.

A *terceira máquina de estados* (4c) envolve transições respeitantes ao cenário da Agregação. Um cliente ao iniciar uma Sessão de Agregação deve indicar os tópicos pretendidos, definir a função de agregação e o comportamento de disseminação dos dados processados. Essa função de agregação pode mais tarde ser substituída. Um utilizador pode também adicionar interações a uma Sessão já existente (segundo o mesmo comportamento) e isso resulta numa Agregação. Deve indicar os tópicos suplementares e definir a função de agregação. Uma Sessão de Agregação torna-se inválida se qualquer uma das suas fontes se tornar indisponível.

A *última máquina de estados* (4d) diz respeito à reconfiguração automática associada ao padrão de Editor/Assinante. O utilizador, quando inicia este tipo de interação, pode indicar adicionalmente um segundo padrão. Logo que o evento associado ao tópico de subscrição é despoletado, a notificação é enviada e o *middleware* reconfigura a Sessão para o dito padrão. Por exemplo, esta transição tem utilidade direta sobre a situação em que a corporação de bombeiros subscreve para valores de temperatura próximos de incêndio e, nesta situação, a interação é automaticamente reconfigurada para agregação de Fluxos Contínuos de Dados sobre variados tópicos.

3.3 API do utilizador

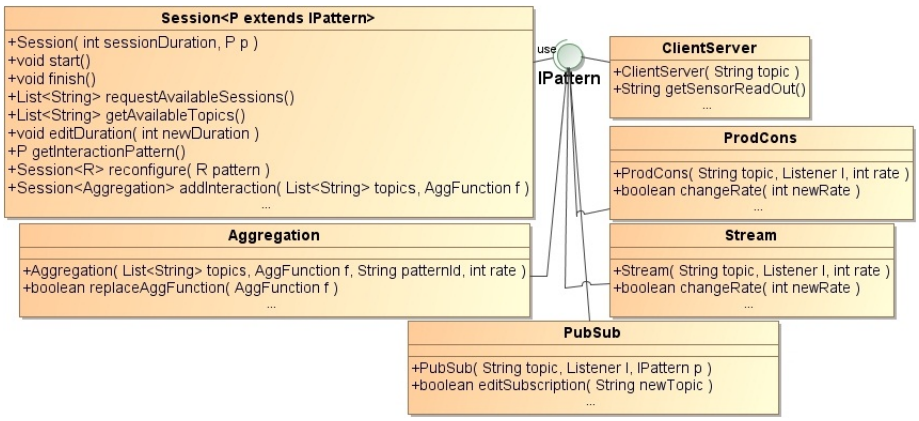


Figura 5: API do utilizador

O diagrama de classes da API do utilizador está ilustrada na Figura 5. Uma Sessão é parametrizada com um padrão. O objeto Session<IPattern> possui métodos respeitantes à manipulação da Sessão. Os padrões (*IPattern*) possuem métodos que apenas fazem sentido no seu modelo de interação e o acesso aos mesmos é dado por Session.getInteractionPattern(). Para os modelos de interação assíncronos, é necessário que o cliente defina um *Listener* que implementa o

método processMessage(MessageNotification m) e que será invocado para o tratamento das mensagens. Da mesma forma, o caso da Agregação implica a definição da função de agregação, que deve implementar o método applyFunction() responsável pela agregação dos dados.

Relembrando o exemplo da deteção de incêndios florestais, a corporação de bombeiros inicia uma Sessão segundo Editor/Assinante sobre o evento "*temperature>50*", como primeiro alerta. Se eventualmente, a temperatura subir para valores críticos (*temperature>70*), o padrão para o qual a sessão é automaticamente reconfigurada será a Agregação dos Fluxos de Dados sobre os tópicos temperatura, direção e velocidade do vento. Esta situação traduz-se para a API do utilizador da seguinte forma:

```
Aggregation agg = new Aggregation( List["Temperature","WindDirection",
                        "WindSpeed"], aggFunction, listener, "Stream", rate);
PubSub critical = new PubSub("Temperature > 70", listener, agg);
PubSub alert = new PubSub("Temperature > 50", listener, critical);
Session<PubSub>  s = new Session<PubSub>(duration,alert);
s.start();
```

4 Trabalho Relacionado

Uma solução de auto-adaptação para os problemas relacionados com o modelo Fluxo de Dados originados num serviço, e.g. associados à perda/atraso de informação decorrentes de perturbações na rede de comunicação, é apresentada em [10]. A solução usa uma estrutura hierárquica distribuída de controladores/atuadores tal que: a) ajusta o fluxo de dados de acordo com as variações dinâmicas detetadas; b) guarda esses dados em memórias tampão sempre que necessário, de modo a evitar perdas de informação; c) processa os dados em trânsito nos nós da hierarquia, reduzindo o tempo de execução da aplicação geradora do fluxo de dados. No entanto, os modelos de interação não aparecem aí como opções explícitas de configuração na interação com um serviço – embora a abordagem acima implemente de facto um modelo de interação (sofisticado) Produtor/Consumidor, tal restringe-se ao sistema de suporte (i.e. não é visível na interação ponto a ponto entre o serviço e o seu utilizador). Para além disso, nada é dito em [10] sobre a possibilidade de, dinamicamente, adicionar novos recetores do fluxo de dados, o que é possível na nossa solução pela junção de novos clientes à respetiva Sessão. Esta permite também que sejam adicionados mais fluxos de dados no contexto da opção de agregação, atrás descrita. Finalmente, o conjunto de modelos de interação disponibilizados na nossa solução é mais rico, e.g. Editor/Assinante com possibilidade de reconfiguração automática para outro padrão após notificação aos assinantes.

Noutras abordagens, o conceito de *padrão* é usado em mecanismos de auto-adaptação de sistemas, mas de modo diferente da nossa solução – e.g. existem trabalhos que utilizam Padrões Arquiteturais reconfiguráveis na definição desses sistemas [11]. Exemplos: o Editor/Assinante permite que se varie quer o número de assinantes, quer os eventos que eles subscrevem (esta característica é também aqui usada no contexto de uma Sessão); o Mestre/Escravo permite que sejam

adicionados novos escravos de modo a otimizar a execução de uma tarefa [12]. No entanto, embora esses trabalhos definam arquiteturas reconfiguráveis, não implementam a noção de Sessão, nem a evolução do sistema com base numa sequência de padrões (e com regras de transição bem definidas).

Finalmente, a solução aqui proposta baseia-se em [9], mas esse trabalho não incide sobre o domínio dos serviços com estado, não tem o conceito de Sessão, nem implementa a evolução do sistema de acordo com uma máquina de estados dependente do contexto de interação entre um serviço e os seus utilizadores.

5 Conclusões e Trabalho Futuro

Tendo em vista permitir modelos de interação mais ricos com RSSF acessíveis por serviços *Web*, foi discutida a relevância da utilização de Padrões de Desenho no desenho de um *middleware* que permite também que essas novas formas de interação possam variar dinamicamente e de acordo com o contexto subjacente aos utilizadores do serviço. Para tal, foi definido o conceito de Sessão que identifica todos os utilizadores que interagem com um serviço particular e com um modo de interação particular. Este pode variar dinamicamente por pedido explícito de um seu utilizador, ou automaticamente de modo a garantir alguns mecanismos de consistência (e.g. por resposta a alterações no contexto de um utilizador). Como trabalho futuro, está já em desenvolvimento a extensão do *middleware* de modo a contemplar a composição de serviços sob a forma de *workflows*, bem como a sua reconfiguração dinâmica.

Agradecimentos: Este trabalho foi parcialmente financiado pelo PEst-OE/EEI/UI0527/2011 Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012.

Referências

1. Botts, M.E., Percivall, G., Reed, C., Davidson, J.: OGC sensor web enablement: Overview and high level architecture. In Nittel, S., Labrinidis, A., Stefanidis, A., eds.: GeoSensor Networks, Second International Conference, GSN 2006, Boston, MA, USA, October 1-3, 2006, Revised Selected and Invited Papers. Volume 4540 of Lecture Notes in Computer Science., Springer (2008) 175–190
2. Paulino, H., Santos, J.R.: A middleware framework for the web integration of sensor networks. In: Sensor Systems and Software - Second International ICST Conference, S-CUBE 2010, Revised Selected Papers. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Springer-Verlag (2011) 75–90
3. Campbell, A.T., Eisenman, S.B., Lane, N.D., Miluzzo, E., Peterson, R.A., Lu, H., Zheng, X., Musolesi, M., Fodor, K., Ahn, G.S.: The rise of people-centric sensing. IEEE Internet Computing **12** (July 2008) 12–21
4. ITU: Itu internet report 2005: The internet of things. Technical report, International Telecommunication Union (2005)
5. Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W., Weerawarana,

S.: Modeling stateful resources with web services v. 1.1. Technical report, Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and The University of Chicago (2004)
6. OASIS: Oasis web services resource framework (WSRF) TC. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`
7. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: a system of patterns. Volume 1. John Wiley and Sons (1996)
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
9. Gomes, C., Rana, O.F., Cunha, J.: Extending grid-based workflow tools with patterns/operators. Int. J. High Perform. Comput. Appl. **22** (August 2008) 301–318
10. Bhat, V., Parashar, M., Kh, M., K, N., Klasky, S.: A self-managing wide-area data streaming service using model-based online control. In: in Proc. 7 th IEEE Int. Conf. on Grid Computing. (2006) 176–183
11. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing degrees, models, and applications. ACM Comput. Surv. **40** (August 2008) 2–25
12. Aldinucci, M., Danelutto, M., Kilpatrick, P.: Towards hierarchical management of autonomic components: A case study. In: Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2009, IEEE Computer Society (2009) 3–10