UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Adaptação Dinâmica de Pilhas de Protocolos Estimulada por Alterações no Contexto

Liliana Wu Freitas Rosa

MESTRADO EM INFORMÁTICA

Setembro 2006

Adaptação Dinâmica de Pilhas de Protocolos Estimulada por Alterações no Contexto

Liliana Wu Freitas Rosa

Dissertação submetida para obtenção do grau de
MESTRE EM INFORMÁTICA

pela

FACULDADE DE CIÊNCIAS DA UNIVERSIDADE DE LISBOA

DEPARTAMENTO DE INFORMÁTICA

Orientador:

Luís Eduardo Teixeira Rodrigues

Júri

Nuno Manuel Ribeiro Preguiça
António Casimiro Costa
Maria Antónia Bacelar da Costa Lopes

Setembro 2006

Este trabalho foi parcialmente suportado pela FCT com o co-finaciamento do FEDER através do Laboratório LaSIGE (POSI/ISFL/13/408) e do projecto MICAS — Middleware para Sistemas Adaptáveis ao Contexto — (POSC/EIA/60692/2004).

We must have talked a little, but I can't, for the life of me recall what about.

zu meiner Familie, den Freunden und meinem Jongleur

Agradecimentos

Gostaria de agradecer a todas as pessoas que me ajudaram, apoiaram e contribuíram na realização deste trabalho. Em primeiro, quero agradecer ao meu orientador, o Professor Luís Rodrigues, pela sua tenacidade, paciência, sugestões, discussões e tempo que disponibilizou para a evolução deste trabalho, e sem dúvida a sua qualidade final. Pelas mesmas razões, quero agradecer à Professora Antónia Lopes todo o esforço dedicado a este trabalho e em futuros, espero. Tenho de agradecer também aos meus colegas do grupo Dialnp pelo apoio e ajuda prestados.

Agradeço também a todos os meus colegas do LaSIGE pelo convívio, boa disposição, críticas construtivas e momentos ímpares desde que trabalho no laboratório. Os meus melhores agradecimentos às pessoas especiais, vocês sabem quem são. Queria ainda extender os meus agradecimentos aos professores do Departamento de Informática pelo apoio e conhecimento partilhado.

Resumo

Esta dissertação propõe uma linguagem para a definição de políticas orientada à adaptação, ciente do contexto, de pilhas de protocolos. Uma metodologia orientada a políticas facilita o desenvolvimento e manutenção das aplicações, permitindo ainda a reutilização do suporte adaptativo em contextos distintos. A abordagem ciente do contexto, com base na monitorização da informação relevante, permite uma resposta automática às alterações detectadas.

A linguagem pressupõe uma arquitectura genérica para o suporte à construção de sistemas adaptativos. Esta arquitectura contempla a monitorização do contexto e o processo de adaptação. O enriquecimento da plataforma *Appia*, resultante na extensão *RAppia*, oferece a separação de aspectos funcionais e adaptativos necessária, bem como uma biblioteca de protocolos extensa para a construção da arquitectura.

Foi concretizado um protótipo da arquitectura genérica, validado através de um caso de estudo específico, sendo apresentada a concretização da política de adaptação e do modelo de contexto. Para o protótipo foram ainda desenvolvidas estratégias para aplicar as acções necessárias à adaptação. A exequibilidade destas estratégias foi ainda avaliada através da medição do intervalo de tempo que dura a adaptação, para cada estratégia, no caso de estudo concretizado.

Palavras Chave: Adaptação, Reconfiguração Dinâmica, Políticas, Contexto, Protocolos de Comunicação, Pilhas de Protocolos

Abstract

This thesis addresses the problem of supporting context-aware dynamic adaptation of protocol stacks through a policy-oriented approach. Such approach promotes the separation of adaptation from protocol logic, thus allowing an easier development, and maintenance of applications. Moreover, this separation promotes the reuse of adaptation logic in other contexts. A context aware approach allows an automatic response to detected context changes.

The described approach relies on a specification language for adaptation policies. Such policies are supported by a generic architecture for adaptive systems composition. Moreover, this architecture relies on context monitoring and adaptation mechanisms. The extension of *Appia* framework, called *RAppia*, offers the required separation of concerns, and a diverse protocol suite to achieve the imperative needs for the architecture construction.

Additionally, an architecture prototype was developed, validated by a specific case study. The case study demanded the specification of an adaptation policy and the development of context models for information monitoring. Along with these needs several strategies for adaptation were also developed. These strategies were evaluated according to the adaptation process duration.

Keywords: Adaptation, Dynamic Reconfiguration, Policy, Context, Communication Protocols, Protocol Stack

Conteúdo

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo é apresentado o tema do trabalho realizado. A motivação, os conceitos nucleares e o contexto do trabalho são analisados sucintamente e são descritos os contributos e resultados obtidos. Por fim, é traçada a estrutura desta dissertação.

1.1 Motivação

Qualquer aplicação distribuída é concebida para ser executada num determinado ambiente, constituído por diversos elementos tais como os dispositivos físicos (processador, memória, etc), *software* de suporte (sistema operativo, *middleware*), rede de comunicações (topologia, largura de banda, taxa de erros, etc), padrões de carga, requisitos do utilizador final (latência tolerada, nível de segurança, etc), entre outros. Nesta dissertação, designa-se o conjunto de componentes que influencia o comportamento da aplicação distribuída por *contexto* de execução.

Como é natural, o desempenho ou mesmo a correcção de uma dada aplicação distribuída depende fortemente do contexto de execução. Desta forma, a actividade de configurar a aplicação, adequando o seu funcionamento às características do contexto, é um aspecto central no desenvolvimento de aplicações distribuídas.

Em ambientes onde o contexto é estável, controlado e homogéneo, a configuração e optimização da aplicação pode ser realizada estaticamente, tipicamente antes da aplicação ser instalada nos dispositivos onde irá executar. Porém, quando o

ambiente é dinâmico, pode ser necessário reconfigurar o sistema em tempo de execução, adaptando-o em função das alterações que ocorrem no contexto. Quando as condições do ambiente variam com frequência é essencial que a reconfiguração seja realizada de forma automática pelo próprio sistema, ou seja, que os sistemas sejam adaptativos.

O desenvolvimento dos sistemas adaptativos é um problema inerentemente complexo. A dificuldade do problema tem sido exacerbada pelos desenvolvimentos tecnológicos recentes. Ao longo dos últimos anos a diversidade de dispositivos, ambientes de execução e condições de utilização das aplicações distribuídas tem vindo a aumentar. Um exemplo paradigmático desta realidade é a emergência dos sistemas móveis e *ad hoc*, caracterizados por uma multiplicidade de dispositivos com diferentes recursos e capacidades computacionais, bem como por redes com topologias dinâmicas e conexão intermitente, cujo número de nós pode variar frequentemente. Além disso, a complexidade das aplicações tem vindo a aumentar progressivamente, sendo hoje construídas com base em múltiplos componentes de *software*, interligados de diversas formas.

Para abordar esta complexidade é necessário recorrer a metodologias adequadas que facilitem o desenvolvimento de aplicações adaptativas e cientes do contexto. Em particular, esta dissertação baseia-se na observação de que uma estratégia adequada para lidar com esta complexidade consiste em separar a adaptação dos aspectos funcionais da aplicação, através de *políticas*. Para além disso, considera-se que a utilização de linguagens para a definição de políticas de adaptação facilita a especificação, análise e reutilização destas políticas.

Atendendo à abrangência do problema em causa, impossível de abordar na sua totalidade no âmbito de uma dissertação de mestrado, este trabalho centra-se num subconjunto particular dos componentes de uma aplicação distribuída: os protocolos de comunicação. Em particular, a dissertação considera sistemas de comunicação que são construídos através da composição de protocolos e que podem ser reconfigurados através de operações como a remoção, adição ou permuta de protocolos. A adaptação dos sistemas de comunicação é estudada sob uma perspectiva orientada a políticas, sensível à informação de contexto, conseguida através da reconfiguração dinâmica de composições de protocolos.

1.2 Contribuições e Resultados

A dissertação tem como procedência a experiência acumulada com a plataforma de suporte à composição e execução de protocolos de comunicação *Appia* (Miranda *et al.*, 2001), desenvolvida na Faculdade de Ciências da Universidade de Lisboa. Com base na análise dos padrões de utilização da *Appia* em diferentes contextos, esta dissertação propõe uma linguagem de especificação de políticas adequada a este domínio em particular. A principal contribuição deste trabalho consiste precisamente na identificação dos estímulos da adaptação, no conjunto de acções de reconfiguração relevantes e nas classes de estratégias usadas para aplicar a reconfiguração neste domínio concreto, facetas estas que se reflectem na linguagem de especificação proposta.

O trabalho realizado nesta dissertação pressupõe uma arquitectura genérica para o suporte à construção de sistemas adaptativos, definindo uma linguagem de especificação de políticas, essencial para a construção de componentes. Esta arquitectura possui um sistema de caracterização e captura de contexto, que alimenta a execução da política, e um sistema de componentes reconfiguráveis, assim como um conjunto de estratégias de reconfiguração que são activadas pela política. Esta arquitectura é utilizada para desenvolver um protótipo que ilustra a aplicação da linguagem proposta num caso concreto de utilização.

Este trabalho produziu os seguintes resultados:

- Uma linguagem de definição de políticas de adaptação para sistemas de comunicação;
- Um protótipo de uma arquitectura que suporta a adaptação do sistema de comunicação estimulada por alterações ao contexto;
- Finalmente, com base na experiência acumulada no desenvolvimento dos resultados anteriores, resultou a identificação de um conjunto de alterações a fazer ao sistema *Appia* com a finalidade de simplificar a utilização de uma metodologia de desenvolvimento de sistemas adaptativos baseada na separação entre as políticas de adaptação e a lógica dos protocolos que constituem o sistema de comunicação.

1.3 Estrutura da Dissertação

O Capítulo 2 visa apresentar os conceitos nucleares ao trabalho e uma panorâmica geral dos vários temas relacionados. São apresentados sistemas de suporte à adaptação e protocolos adaptativos. O Capítulo 3 descreve a arquitectura de suporte à construção de sistemas adaptáveis, apresentando os componentes e a dinâmica entre os mesmos. O Capítulo 4 visa a declaração de políticas de adaptação, descrevendo a linguagem que permite a especificação das políticas. O Capítulo 5 apresenta a concretização da arquitectura proposta. A arquitectura é avaliada no Capítulo 6, recorrendo a um caso de estudo. Finalmente o Capítulo 7 conclui esta dissertação, apresentando uma introspecção sobre a continuidade do trabalho realizado.

Capítulo 2

Panorâmica

Esta dissertação aborda o problema da adaptação ao contexto em protocolos de comunicação. Neste capítulo são introduzidos os principais conceitos subjacentes ao trabalho, assim como as principais estratégias descritas na literatura para concretizar a adaptação neste domínio. Finalmente, são enunciados os principais aspectos a considerar no desenvolvimento de sistemas de comunicação adaptativos.

2.1 Conceitos

Nesta secção são introduzidos dois conceitos fundamentais para a compreensão do trabalho realizado, nomeadamente a noção de *contexto* e de *adaptação*.

2.1.1 Contexto

Genericamente, designamos por *contexto* o conjunto de factores que influencia o comportamento e desempenho de uma determinada aplicação. Desta forma, o contexto agrega factores tão distintos como as características dos dispositivos onde a aplicação se executa, até às preferências do utilizador final. De modo a facilitar a sua análise, é comum organizar a informação de contexto em diferentes categorias (Chen & Kotz, 2000):

- Contexto computacional, abarcando factores como a largura de banda, conectividade da rede, capacidade do processador, etc;

- Contexto do utilizador, abarcando factores que dependem do utilizador da aplicação, como as suas preferências, localização geográfica, tipo de actividade desenvolvida, nós da vizinhança, etc;

- Contexto físico, abarcando factores externos ao sistema como a iluminação, nível de ruído, temperatura, etc.

Relativamente à forma como a informação de contexto é obtida (Acharya *et al.*, 1997), pode distinguir-se, nomeadamente, a captura por solicitação (*on-demand*) e a captura contínua. Na captura por solicitação a informação de contexto é apenas recolhida em resposta a um pedido explícito. Na captura contínua, o sistema monitoriza o contexto de forma automática, tipicamente através de leituras periódicas. A frequência de captura da informação determina o custo da monitorização do contexto. A captura contínua recolhe informação em intervalos de tempo regulares, com uma frequência específica, sendo o seu custo proporcional à frequência de captura. Na captura por solicitação a informação é recolhida no momento em que é solicitada, apresentando um custo incremental, comparativamente mais baixo.

A informação de contexto capturada não é necessariamente informação pronta a ser utilizada, dado que poderá não apresentar o nível de abstracção adequado para as aplicações que dela necessitam. Assim, a informação é tratada e organizada em conformidade, para ser possível disponibilizá-la aos elementos que a requisitem. Desta forma, o processo de obtenção da informação de contexto assenta nos seguintes passos:

1. Capturar informação acerca dos diversos factores que constituem o contexto;

2. Integrar esta informação numa entidade logicamente coerente que agrega a informação de contexto proveniente de diferentes fontes;

3. Representar esta informação de forma estruturada, de modo a facilitar a sua análise;

4. Fornecer uma interface para os restantes componentes do sistema que facilite a activação de acções em resposta a alterações no contexto.

O conjunto de componentes que executa as acções acima descritas designa-se por *monitor de contexto*. A captura de informação de contexto é realizada por componentes que, nesta dissertação, se designam por *sensores de contexto*. A materialização de um sensor de contexto depende do factor que se está a monitorizar. Certas características podem ser monitorizadas invocando interfaces de gestão ou capturando excepções. Outras recorrendo a metodologias de programação como padrões de desenho ou reflexão estrutural, no caso de mecanismos orientados a objectos.

A informação proveniente de diversos sensores deve, posteriormente, ser agregada numa entidade logicamente coerente que facilite a correlação entre a informação de diferentes fontes. Num sistema distribuído a forma mais fácil de conseguir esta integração consiste em armazenar a informação num repositório centralizado. Neste caso a informação recolhida pelos sensores deve ser transportada para o repositório. Isto pode ser realizado utilizando a propagação assíncrona de eventos ou a propagação periódica de mensagens de controlo, por vezes também designadas por *pulsações* (*heartbeats*). Num sistema distribuído complexo, é possível que, em cada nó do sistema, só seja necessário aceder a um subconjunto da informação de contexto disponível. Neste caso, um sistema de edição-subscrição (Eugster *et al.*, 2003) pode ser o mais adequado para transportar a informação dos sensores até aos nós que necessitam de tratar essa informação.

Após recolhida, a informação de contexto é armazenada para posterior consulta e análise. Dependendo da sofisticação do sistema de contexto e do tipo de cenário em que a informação de contexto é utilizada, a informação proveniente dos sensores pode ser processada de diferentes formas antes de ser disponibilizada. Nos casos mais simples, a informação recolhida pelos sensores pode ser utilizada directamente para activar a adaptação. Por exemplo, um protocolo de transporte pode adaptar-se em função da taxa de erros observada no nível rede. Noutros casos, o sistema de contexto pode realizar operações simples sobre os valores capturados pelos sensores, tal como manter um valor médio das últimas leituras. No entanto, nos casos mais ricos, o sistema de contexto pode extrair informação de alto-nível a partir da informação recolhida. Por exemplo, o sistema de contexto pode relacionar os valores de vários sensores de forma a reconhecer e identificar

situações relevantes (Coutaz *et al.*, 2005)("o utilizador está a descansar", "a rede está estável").

Finalmente, o monitor de contexto deve exportar um conjunto de interfaces que facilitem a execução de estratégias adaptativas. Tipicamente, existem dois tipos de interfaces úteis de exportar: síncronas e assíncronas. As interfaces síncronas facilitam a análise da informação de contexto armazenada (fazer interrogações, buscas, estabelecer correlações, etc). As interfaces assíncronas são úteis para assinalar a ocorrência de eventos relevantes (alarmes, excepções, entrada e saída de membros, etc).

2.1.2 Adaptação

Designa-se por adaptação o processo de alteração dinâmica do comportamento do sistema. Este processo, tipicamente, responde a uma alteração no contexto e tem como objectivo preservar um conjunto de propriedades no serviço prestado pelo sistema. Por exemplo, segurança, desempenho, satisfação do utilizador, etc.

Nesta dissertação, adopta-se uma perspectiva lata da definição de contexto, tal como referido na secção anterior. Daí se considerar que os diversos factores que estimulam a adaptação podem ser capturados e representados pelo monitor de contexto, ou seja, nesta dissertação, considera-se que a adaptação é estimulada sempre por alterações ao contexto. Esta simplificação facilita a concepção e descrição do sistema. Convém no entanto sublinhar que em muitos sistemas o contexto não se materializa de forma explícita. Por exemplo, a adaptação pode ser estimulada por directivas explícitas do utilizador, caso em que o monitor de contexto se limita a memorizar os comandos do utilizador. Noutros casos, a adaptação é estimulada pelo tipo de tarefa que o sistema está a realizar. Por exemplo, o sistema de comunicação pode adaptar-se ao tipo de conteúdo da informação que é transportada, utilizando mecanismos diferentes para transportar dados, vídeo ou áudio. Este tipo de cenário é por vezes designado por adaptação estimulada pelo conteúdo, uma vez que que a informação de contexto relevante para a aplicação é apenas o conteúdo da informação processada pelo sistema.

Em relação à forma como se define a lógica de adaptação importa referir que, num sistema adaptativo, existe sempre um conjunto de regras, implícito ou

explícito, que captura a lógica da adaptação. Estas regras avaliam o contexto e definem quais as acções a tomar em caso de alteração no mesmo. A lógica da adaptação tem subjacente um objectivo e um plano para atingir este objectivo. Desta forma, pode-se dizer que a lógica da adaptação materializa sempre uma *política de adaptação*, mesmo quando esta política não é capturada de forma explícita. O conjunto de regras, quando explícito, pode ser fixo ou, ele próprio, ser alterado dinamicamente, através da alteração ou substituição das regras em tempo de execução (Montanari *et al.*, 2004).

A forma como a política de adaptação é codificada é um dos factores que distinguem os sistemas adaptativos. As abordagens variam entre um modelo de adaptação embebido e desacoplado. No primeiro caso, não é possível oferecer uma separação clara entre os aspectos funcionais e adaptativos. No segundo caso, a separação destas facetas é clara. Estas diferenças têm impacto a nível do custo de desenvolvimento do sistema, da manutenção e da especificação da adaptação.

No modelo de adaptação embebida, a lógica adaptativa encontra-se imbricada de forma estática (*hard-coded*) na lógica funcional (Sudame & Badrinath, 2001). Em sistemas que utilizam este modelo, o mesmo código que executa as funções da aplicação realiza também a monitorização do contexto e a adaptação, sempre que necessário, sem auxílio de componentes adicionais. Como todas as abordagens monolíticas, este modelo é de desenvolvimento mais expedito e oferece à partida um melhor desempenho mas torna o código da adaptação bastante difícil de manter e de reutilizar.

No modelo de adaptação desacoplada o código da adaptação é separado do código que executa os componentes funcionais da aplicação. Esta separação clara permite obter uma lógica de adaptação modular, fácil de actualizar ou substituir sem serem necessárias alterações a nível funcional. Esta abordagem permite também a reutilização da lógica adaptativa noutros sistemas. Por outro lado, esta abordagem obriga à criação de interfaces de controlo, que permitam ao componente de adaptação monitorizar e alterar a execução do componente funcional. Para além das vantagens oferecidas pela separação de aspectos, a utilização de políticas de adaptação permite a definição de padrões de políticas para especificar estratégias comuns e reutilizar essas estratégias noutras aplicações.

O desacoplamento do código da adaptação do código funcional pode ser feito em diferentes graus. Em sistemas com vários componentes podemos distinguir duas aproximações. Uma em que adaptação é realizada internamente a cada componente e outra em que a adaptação é realizada por uma entidade exterior aos componentes do sistema (Rigole *et al.*, 2004). A adaptação externa é necessariamente desacoplada e requer a utilização de uma arquitectura onde o ciclo de controlo constituído pela monitorização de contexto, execução da política e reconfiguração, aparece de forma explícita.

Como é natural, a definição da política de adaptação está intimamente dependente dos mecanismos disponíveis para alterar o comportamento do sistema. Deste ponto de vista, é interessante distinguir os sistemas onde conjunto de componentes e as ligações entre estes é fixo, sendo apenas possível afectar o comportamento de cada um dos componentes através da alteração de parâmetros de configuração (McKinley *et al.*, 2004), dos sistemas onde é possível alterar a estrutura do sistema, através da inserção, remoção e substituição de componentes (Magee *et al.*, 1989) e da modificação da forma como estes se interligam (Garlan *et al.*, 2004). Neste último caso a adaptação também se designa por reconfiguração dinâmica.

Os sistemas que utilizam a reconfiguração dinâmica seguem necessariamente uma abordagem externa. Neste sistemas, a forma mais natural de capturar a política de adaptação é através da utilização de linguagens de especificação de políticas, tipicamente baseadas num conjunto de regras do tipo *Evento-Condição-Acção* (McCarthy & Dayal, 1989) cujos campos determinam, respectivamente, quais os factores que activam a regra, quais as condições que se aplicam e qual a acção ou acções a tomar.

2.2 Sistemas de Suporte à Adaptação

Nesta secção são descritos alguns sistemas adaptativos existentes na literatura, que seguem estratégias distintas para a especificação da lógica da adaptação. Estes sistemas ora seguem o modelo de adaptação embebida ou desacoplada, introduzidos na secção anterior.

2.2.1 Adaptação Embebida

Como referido na secção 2.1.2, o modelo de adaptação embebida não oferece uma separação clara dos aspectos funcionais e não funcionais (adaptativos), o que resulta numa abordagem monolítica com vantagens e desvantagens. A principal vantagem deste modelo é oferecer um desempenho superior, comparativamente a um modelo desacoplado. Tal deve-se ao facto de todos os aspectos estarem concretizados no mesmo componente, não existindo atrasos induzidos pelas interfaces de ligação dos diversos componentes. No entanto, esta concretização dificulta a manutenção dos vários aspectos do sistema, que acabam por estar interligados. Alterações em aspectos funcionais podem implicar alterações na lógica da adaptação e vice-versa. A concretização monolítica prejudica ainda a reutilização da parte adaptativa noutros contextos.

A adopção de um modelo de adaptação embebida num sistema pressupõe que as vantagens ultrapassam as desvantagens do modelo. Em muitas situações, opta-se por esta estratégia dada a importância do desempenho. Em alguns casos, esta opção deve-se a necessidades adaptativas simples, não se justificando o desenvolvimento de componentes específicos para a monitorização do contexto e para a gestão da adaptação. Mas podem existir razões diversas. Por exemplo, em situações em que a monitorização do contexto está intimamente ligada aos aspectos funcionais do sistema, uma abordagem monolítica facilita a captura da informação.

Os exemplos mais frequentes de adaptação embebida são os protocolos adaptativos. Estes protocolos fornecem um serviço específico que já tem em consideração as adaptações necessárias. A sua concretização aborda os aspectos funcionais e adaptativos no mesmo bloco. Exemplos destes protocolos, nomeadamente de comunicação, serão abordados na secção 2.3.1.

2.2.2 Adaptação Desacoplada

Tal como a adaptação embebida, a adaptação desacoplada oferece vantagens e desvantagens, derivadas da separação dos aspectos funcionais e não funcionais. Esta separação permite alterar a estratégia de gestão da adaptação sem repercussões na concretização dos aspectos funcionais (Sloman, 1994). Para além disso,

permite a sua reutilização noutros contextos, bem como efectuar alterações dinâmicas sem necessitar de interromper a execução. Contudo, a separação dos aspectos exige um suporte adicional para a gestão automática dos mecanismos relacionados com a adaptação. Mas esse suporte adicional acarreta custos. O desempenho é afectado pelos atrasos introduzidos pela interacção extra entre os componentes responsáveis pelos aspectos funcionais e pelos não funcionais.

Existem vários sistemas que seguem um modelo de adaptação desacoplada na sua abordagem. O modelo permite-lhes usufruir das vantagens referidas para atingir os seus objectivos nos ambientes dinâmicos e heterogéneos a que se destinam. São exemplos o ambiente *Poema*, a linguagem de especificação de políticas *Ponder* e a plataforma *Chisel*, entre outros. Nas secções que se seguem são descritos os exemplos mencionados.

2.2.2.1 Ambiente *Poema*

O *Poema*, desenvolvido por Montanari *et al.* (2003, 2004), é um ambiente de suporte ao desenvolvimento de aplicações de código móvel, com base em políticas. Designa-se por mobilidade de código a capacidade de alterar em tempo de execução as ligações entre diferentes fragmentos de código e a localização onde são executados (Carzaniga *et al.*, 1997). Este paradigma de programação oferece diversas vantagens na estruturação de sistemas distribuídos em ambientes dinâmicos e heterogéneos, descritas por Picco (2000).

O princípio da separação de aspectos, funcionais e não funcionais da aplicação, oferece o suporte necessário à adaptação dinâmica, aumentando também a flexibilidade dos sistemas de software. A abordagem do ambiente *Poema* assenta numa separação entre os aspectos funcionais da aplicação e os requisitos da mobilidade de código. Assim, a modificação dos aspectos não funcionais (requisitos de mobilidade de código) é realizada sem afectar a concretização dos aspectos funcionais, bem como não pressupõe a interrupção da execução da aplicação. Além disso, é possível, através da análise, detectar possíveis incoerências ou outros problemas.

Existem vários métodos para conseguir a separação de aspectos. Uma possibilidade é tornar as políticas de adaptação explícitas. Estas podem ser expressas

através de conjuntos de regras que determinam o comportamento do sistema, separadamente dos pormenores de concretização do mesmo. Dado que o suporte ao desenvolvimento de aplicações de código móvel passa pela definição de estratégias para a migração de código, no *Poema* as políticas são responsáveis pela especificação dessas estratégias. O ambiente *Poema*, concretizado em *Java*, utiliza a linguagem *Ponder* (descrita na secção 2.2.2.2) para especificar as políticas de mobilidade de código.

Com base na abordagem de separação de aspectos seguida, o *Poema* considera que uma aplicação de código móvel é composta por componentes estáticos (*code components*) e dinâmicos (*computational components*). Os primeiros são descrições estáticas de aspectos que podem ser adaptados. Os segundos são fluxos de execução, autónomos, dos diversos componentes estáticos. Uma aplicação é constituída por diferentes componentes dinâmicos que interagem. Os componentes dinâmicos são constituídos pelo estado (informação relativa à aplicação), comportamento da aplicação (concretização da lógica da aplicação) e pelo comportamento de mobilidade (política que define quais as migrações de código mais adequadas para cada situação). Assim, esta separação de aspectos permite a adaptação sob a forma de reconfiguração dinâmica, sem existir um impacto na concretização dos aspectos funcionais da aplicação e sem interromper a execução.

A reconfiguração tem lugar a nível da política e da aplicação. A reconfiguração dinâmica das políticas de adaptação permite a sua substituição em tempo de execução. Assim, é possível carregar dinamicamente políticas com novas estratégias ou eliminar estratégias inadequadas. A interpretação das políticas de adaptação resulta, então, na alteração do conjunto de fragmentos de código que executam. No *Poema*, a reconfiguração dinâmica da aplicação é conseguida através de alterações ao conjunto de componentes de uma aplicação. Os componentes dinâmicos têm a responsabilidade de carregar os componentes estáticos necessários e remover os inadequados, alterando a composição de código da aplicação.

Uma abordagem orientada a políticas necessita de suporte de gestão durante o tempo em que uma política está activa. Assim, a arquitectura do ambiente *Poema* está organizada em duas camadas distintas de serviços: *basic* e *policy*. Esta organização é descrita pela Figura 2.1.

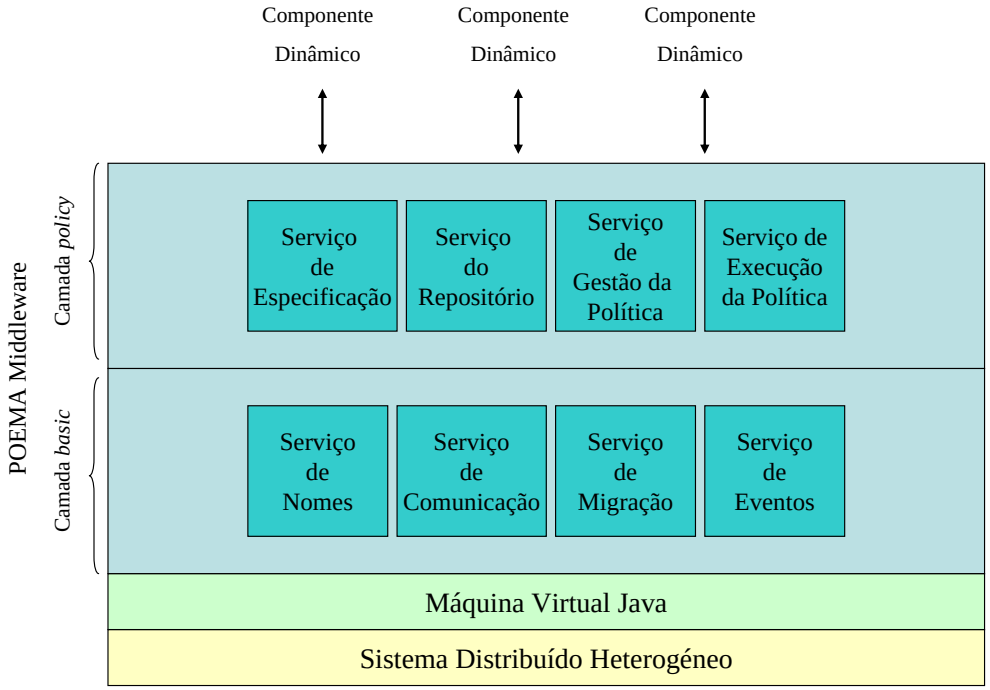

Figura 2.1: Arquitectura do ambiente *Poema*

A primeira camada fornece os serviços base para a aplicação: serviço de nomes, comunicação, migração e de eventos. O serviço de nomes permite associar identificadores únicos a cada componente dinâmico. O serviço de migração é responsável pelos diferentes tipos de mobilidade, como a migração ou a recolocação de componentes estáticos em componentes dinâmicos. O serviço de comunicação fornece as ferramentas necessárias para a coordenação e comunicação de componentes dinâmicos. Por fim, o serviço de eventos permite a monitorização do estado do ambiente de execução da aplicação, notificando as alterações relevantes através de eventos. A informação monitorizada pode ser relativa aos recursos, ao utilizador ou mesmo à aplicação

A segunda camada é responsável pelo suporte do desenvolvimento das políticas *Ponder*, sendo constituído pelos módulos: serviço de especificação, repositório, de gestão e de execução de políticas. O serviço de especificação utiliza as ferramentas da linguagem *Ponder* para executar diferentes operações sobre a política. Existem diversas operações possíveis como a actualização, adição ou remoção de regras da

política. Este serviço cria ainda os objectos *Java* que representam cada política. Estes objectos são registados num repositório próprio que distribui a informação pelos componentes dinâmicos. O repositório é gerido por um serviço dedicado, com o mesmo nome. O serviço de gestão de políticas gere as políticas durante o seu tempo de vida. Este serviço armazena informação relativa, sendo responsável pela sua distribuição. Por exemplo, o serviço pode notificar quando é que uma política deixa de estar activa. Estas notificações são da responsabilidade do serviço de eventos. Por fim, o serviço de execução de políticas aplica a política sempre que ocorrem eventos relacionados com a mobilidade. Este serviço interpreta a política e activa as acções especificadas nas regras que se aplicam.

2.2.2.2 Linguagem *Ponder*

A *Ponder* foi desenvolvida por Damianou *et al.* (2001) e é uma linguagem declarativa, orientada a objectos. Inicialmente, a linguagem foi desenvolvida para a especificação de políticas de segurança e gestão em sistemas distribuídos. A linguagem apresenta-se como uma linguagem geral que visa oferecer uma aproximação unificada à especificação, suportando os conceitos das diferentes políticas emergentes, cumprindo diversos requisitos. Estes requisitos vão deste a flexibilidade e extensibilidade da linguagem, à possibilidade de análise de conflitos e incoerências entre diferentes regras da política, oferecendo uma abordagem para a estruturação e composição de regras.

A linguagem permite especificar diversos comportamentos para cada objecto. Cada regra define a escolha do comportamento mais adequado para um objecto, em circunstâncias bem definidas. Os eventos permitem sinalizar as novas circunstâncias às quais é necessário reagir. Por exemplo, uma tentativa de *login* falhada três vezes. Os eventos podem assinalar alterações a nível interno ou externo, detectadas por serviços de monitorização. Cada regra é despoletada por um evento e segue uma estrutura condição-acção. As condições são predicados que necessitam de ser avaliados para determinar se a regra é aplicável ou não. Por fim, as acções definem qual o novo comportamento do objecto ou objectos alvo.

A sintaxe de uma regra assenta em quatro primitivas: *subject*, *target*, *action* e *when*. A primeira primitiva, *subject*, define a responsabilidade de gestão, a cargo

de utilizadores, gestores ou componentes. A segunda primitiva, *target*, determina quais os objectos alvo das acções, por exemplo recursos ou serviços. A terceira primitiva, *action* especifica quais as acções a realizar. A quarta e última primitiva, *when*, determina restrições à aplicação da regra.

O objectivo das regras separa-as em tipos. Assim, existem cinco tipos possíveis: autorização (*authorisation*) , delegação (*delegation*), filtragem de informação (*information filter*), moderação (*refrain*) e obrigação (*obligation*). O tipo autorização determina que actividades podem ser realizadas, controlando o acesso a serviços e recursos. As regras podem permitir (positivas) ou proibir (negativas) o acesso aos objectos. O tipo moderação define quais as actividades que são proibidas. O seu objectivo é semelhante ao das regras de autorização negativa, no entanto existe uma distinção clara. As regras de moderação são postas em vigor pelo *subject*, ao passo que nas regras de autorização essa responsabilidade cabe ao *target*. O tipo filtragem de informação refere-se apenas às regras de autorização positiva, controlando se a informação de retorno do resultado das acções, pode ser apresentada directamente ou necessita de ser transformada previamente. O tipo delegação é utilizado para transferir temporariamente direitos de acesso de um utilizador para outro. Por fim, as regras do tipo obrigação permitem especificar que acções devem ser realizadas em determinadas situações e por quem. Estas regras permitem especificar alterações de comportamento, adequando-o à situação actual do sistema.

Como referido, a sintaxe das regras assenta no mesmo conjunto de primitivas. No entanto, as necessidades especificas de cada tipo, introduzem novas primitivas. A título de exemplo é apresentada a sintaxe do tipo autorização:

```
inst ( auth+ | auth- ) ruleName "{"
subject [<type>] domain-Scope-Expression ;
target [<type>] domain-Scope-Expression ;
action action-list ;
[ when constraint-Expression ; ] "}"
```

A sintaxe acima apresenta as primitivas comuns: *subject*, *target*, *action* e *when*. Para além disso, apresenta a opção, específica a este tipo de regra, relativa à autorização ser positiva (*auth+*) ou negativa (*auth-*).

A linguagem *Ponder* permite ainda definir meta-regras (regras acerca de regras) e composições de regras com um domínio (*domain*) comum, no entanto é omissa em relação à especificação dos eventos e notificações relevantes. Apesar de ter sido desenvolvida para políticas de segurança, a linguagem adequa-se à definição de políticas em geral.

2.2.2.3 Plataforma *Chisel*

A *Chisel*, desenvolvida por Keeney & Cahill (2003), é uma plataforma de suporte à adaptação dinâmica de objectos através de uma abordagem ciente do contexto e baseada em políticas. A plataforma é orientada às necessidades da adaptação não antecipada, permitindo alterações ao comportamento em tempo de execução. A *Chisel*, tal como o ambiente *Poema*, segue uma abordagem de separação de aspectos funcionais e não funcionais. A separação dos aspectos permite especificar os vários comportamentos possíveis (relacionados com os aspectos não funcionais) de forma independente da concretização dos aspectos funcionais da aplicação. Esta abordagem oferece várias vantagens, já descritas na secção 2.2.2.1. No entanto, a separação de aspectos na plataforma *Chisel* é conseguida de forma diferente do ambiente *Poema*. A plataforma segue uma abordagem reflexiva. O papel das políticas na plataforma será abordado mais à frente.

A separação de aspectos permite levantar os vários aspectos não funcionais de cada objecto. De acordo com esses aspectos é possível definir quais os comportamentos que um objecto pode ter. A reflexão representa esses comportamentos através de meta-tipos. A associação entre os comportamentos e os objectos pode ser alterada em tempo de execução, determinando um comportamento diferente do objecto. A plataforma *Chisel* oferece a adaptação dinâmica através de alterações das associações objecto e comportamento, bem como do conjunto de comportamentos disponíveis para cada objecto. Estas alterações não pressupõem a interrupção da execução nem a alteração da concretização dos aspectos funcionais dos objectos.

A capacidade de reflexão resulta num nível de abstracção adicional do sistema. Assim, um sistema tem dois níveis: nível-base e meta-nível. O nível-base inclui os objectos serviço, a aplicação e a política de adaptação. O meta-nível agrupa

os comportamentos de cada objecto e o gestor da adaptação, responsável também pela monitorização da informação de contexto. A informação de contexto monitorizada é relativa ao utilizador, à aplicação e ao ambiente de execução.

A plataforma *Chisel* baseia-se na constatação de que o utilizador e a aplicação possuem informação importante para ditar a adaptação. Dado que se tratam de duas entidades distintas, a sua colaboração é conseguida através das políticas de adaptação. A política é responsável pelo processo de decisão, determinando qual o comportamento mais adequado nas condições monitorizadas. A plataforma *Chisel* dispõe de uma linguagem própria para a definição das regras de decisão da política, bem como dos eventos e notificações relacionados com a informação de contexto. A linguagem é abordada mais à frente.

A interpretação da política de adaptação é activada pelo gestor da adaptação, responsável por monitorizar e notificar as alterações de contexto relevantes. O resultado da interpretação dita qual o comportamento adequado e o gestor é responsável por associar o novo comportamento ao objecto.

O gestor da adaptação é constituído por vários gestores. Um desses gestores é o do contexto, que através do serviço de eventos, sinaliza alterações nos recursos, utilizador ou aplicação. Existem gestores responsáveis pela política de adaptação, que mantêm um histórico das alterações à política de adaptação e são responsáveis pela interpretação das regras da política. Por fim, existe ainda um gestor responsável por aplicar as alterações ditadas pelas regras da política que controla o conjunto de comportamentos e as associações com os objectos de serviço.

A plataforma *Chisel*, ao contrário do ambiente *Poema*, não oferece reconfiguração dinâmica, mas sim *customization*(personalização) dinâmica (Montanari *et al.*, 2003). A distinção está no nível a que as alterações ocorrem. Na reconfiguração dinâmica, no caso da mobilidade de código, existe uma alteração da composição de fragmentos de código a executar, o que ocorre no *base-level*. No caso da personalização dinâmica, o código em execução não é alterado, apenas os meta-tipos que estão associados são substituídos por outros. A personalização, apesar de também acontecer dinamicamente, não implica a mesma complexidade da reconfiguração dinâmica, nem uma alteração efectiva dos objectos.

Linguagem de Definição de Políticas da *Chisel* A linguagem de definição de políticas da plataforma é orientada às adaptações comportamentais e estruturais de grau simples. A linguagem permite definir regras e eventos.

Cada regra da política determina quais as alterações de comportamento/estrutura a efectuar no sistema gerido. Uma regra é constituída por três partes: despoletar, acção e restrições de aplicação. A primeira parte define quais os eventos (alterações) que accionam a regra, especificada pela primitiva *ON*. A segunda parte determina qual a resposta aos eventos, ou seja, quais as acções de adaptação a realizar. A terceira parte é relativa a restrições ou salvaguardas que possam existir, especificadas pela primitiva *IF*. A sintaxe referida é a seguinte:

```
ON: evento
        acção
                IF: condição
```

Esta linguagem, ao contrário da *Ponder*, não permite especificar âmbitos de acção nas regras, ou seja, não existe suporte para distinguir um subconjunto de entidades afectadas. Para além disso, não é possível definir quem é responsável por executar as alterações. Por fim, dada a simplicidade da adaptação, não existem tipos distintos de regras, logo a sintaxe é única.

Na linguagem, cada evento sinaliza alterações de contexto relevantes ou, então, o retorno de uma adaptação. Os eventos podem ser definidos em tempo de execução através de uma primitiva *NEW* que aceita o nome do evento e qual a condição ou condições que se aplicam para ser accionado. A sintaxe é a seguinte:

```
NEW EventTrigger Condição
```

2.3 Adaptação de Sistemas de Comunicação

A qualidade e o desempenho da comunicação dependem de vários aspectos relacionados não só com as necessidades das aplicações mas também com as características da rede. A adaptação da comunicação permite optimizá-la ou adequar as suas propriedades.

A abordagem à adaptação nos sistemas de comunicação varia em função do modo como estes são construídos. A construção pode tomar uma forma monolítica, em que um único componente concretiza todos os protocolos necessários,

ou através de uma composição de vários componentes, cada um concretizando protocolos distintos. No segundo caso diz-se que o sistema suporta a composição de protocolos. Cada abordagem tem as suas vantagens. A abordagem monolítica oferece um desempenho superior à abordagem modular, no entanto, a última é preferível para a manutenção e reutilização de protocolos.

Os sistemas de comunicação podem, tal como os sistemas de suporte à adaptação, seguir um modelo embebido ou desacoplado. O modelo embebido refere-se a protocolos com a capacidade para se auto adaptarem, oferecendo comportamentos distintos no mesmo protocolo. Estes protocolos denominam-se *protocolos adaptativos*. O modelo desacoplado refere-se à adaptação das composições de protocolos. O modelo de composição mais comum utiliza composições verticais, também designadas por pilhas de protocolos. Neste caso, a reconfiguração é conseguida através das operações de remoção, adição ou substituição de protocolos

Nas secções que se seguem são abordados os protocolos adaptativos e as molduras de suporte à composição de protocolos, focando exemplos das diferentes abordagens.

2.3.1 Protocolos Adaptativos

Os protocolos adaptativos resultam da necessidade de lidar com a diversidade crescente de ambientes dinâmicos e heterogéneos. As vantagens da adaptação impulsionaram um número crescente destes protocolos. Os protocolos adaptativos apresentam-se ora como protocolos com suporte à adaptação concebido de raíz, ora como versões adaptativas de protocolos já existentes. Em qualquer dos casos, a maioria dos protocolos adaptativos são orientados à optimização do desempenho, para condições bem definidas.

As versões adaptativas de protocolos para redes móveis são frequentes. Os protocolos têm de lidar com perturbações constantes na comunicação, dado o seu dinamismo, pelo que uma abordagem adaptativa permite optimizar o desempenho consideravelmente. Existem versões adaptativas de protocolos de transporte, como o *TCP* e *UDP* (Xylomenos & Polyzoz, 1999), de encaminhamento (Heinzelman *et al.*, 1999; Park & Corson, 1997), de difusão (Gupta & Srimani, 1999), entre

muitos outros. Na secção seguinte aborda-se um desses protocolos adaptativos, o *TCP*.

2.3.1.1 Protocolo *TCP*

O protocolo *TCP* (Postel, 1981) oferece um serviço de comunicação ponto-a-ponto, orientado à ligação, que assegura a entrega fiável e ordenada dos dados, assim como funções de controlo de fluxo e congestão. As funções de controlo embebidas no protocolo são optimizadas para redes fixas e baseiam-se em diferentes parâmetros que reflectem as condições da rede.

As condições da rede são reflectidas pela percentagem de *acknowledgements* (*acks*) recebidos e por temporizadores (*timeout*). Os *acks* permitem ajudar a caracterizar o estado de congestão da rede: quanto menor a percentagem de *acks* recebidos, maior é a congestão. Os temporizadores permitem detectar a perda de segmentos: quanto maior o número de retransmissões devidas à expiração do temporizador, maior a congestão.

É com base na caracterização das condições da rede que o protocolo se reconfigura. A reconfiguração ocorre em tempo de execução, a nível do valor do temporizador e do tamanho da janela de congestão. A afinação destes parâmetros permite adequar o fluxo de informação às condições da rede e também à capacidade de processamento de dados no receptor.

O valor do temporizador possui um papel fundamental no desempenho do protocolo. Um valor demasiado pequeno obriga a transmissões desnecessárias e um valor demasiado elevado atrasa a recuperação de pacotes perdidos. A afinação do temporizador depende da topologia da rede, bem como da carga a que esta está sujeita em cada instante. Assim, o valor do temporizador pode variar ao longo do tempo de vida de uma ligação *TCP*. O *TCP* inclui mecanismos para aferir qual o melhor valor a utilizar e adapta-o consoante as condições da rede.

O tamanho da janela de congestão determina a capacidade da rede. O protocolo apenas transmite novo bloco de informação após o *ack* relativo à recepção do bloco previamente transmitido. A afinação do tamanho da janela de congestão permite controlar o fluxo de informação, com o objectivo de equilibrar a carga da rede e lidar com a congestão.

O desempenho do *TCP* em redes sem fios, para as quais não foi optimizado, cai consideravelmente. As optimizações acima referidas chegam a ser prejudiciais, degradando ainda mais a qualidade da comunicação. O controlo de congestão do protocolo *TCP* interpreta a perda de informação resultante das flutuações, ocasionais mas frequentes, da largura de banda ou *handoffs*, como congestão da rede. Esta suposição prejudica a comunicação, tornando-a mais lenta. Por essa razão, existem versões do protocolo *TCP* orientadas a redes sem fios que apresentam desempenhos superiores (Xylomenos & Polyzoz, 1999).

Nas redes móveis existe uma grande interacção entre o funcionamento do protocolo de transporte e o funcionamento dos protocolos do nível de ligação de dados, nomeadamente dos protocolos da camada de controlo de acesso ao meio MAC (*Media Access Control*) (Kwon *et al.*, 2004)). Por exemplo, informação desta camada pode ser útil para escolher qual a melhor estratégia para recuperar omissões no nível de transporte. Esta interacção é complexa quando os vários protocolos aplicam estratégias adaptativas independentes e não coordenadas. Por esta razão, alguns autores defendem que é necessário criar protocolos que abordam facetas transversais em relação ao modelo em camadas tipicamente usado na concepção de sistemas de comunicação, como é o caso das abordagens *cross-layer design* (Kawadia & Kumar, 2003).

2.3.2 Molduras de Suporte à Composição de Protocolos

Uma moldura de suporte à composição de protocolos oferece uma abordagem desacoplada, suportada por um ambiente de execução, um conjunto de bibliotecas que suportam o modelo de composição e um conjunto de protocolos. O modelo de composição descreve a interface a que os protocolos devem obedecer e o modo como podem ser compostos para criar serviços de comunicação. Relativamente aos protocolos, cada um é desenvolvido tendo em conta apenas as propriedades que oferece, sem considerar qualquer composição, facilitando o desenvolvimento e avaliação de cada protocolo. A utilização destas molduras possibilita a adaptação através da reconfiguração dinâmica.

Existem diversas plataformas de composição de protocolos, entre elas o *Coyote*, o *Cactus* e o *Appia*.

2.3.2.1 Coyote e Cactus

As plataformas *Coyote* (Bhatti *et al.*, 1998) e *Cactus* (Hiltunen *et al.*, 2000) partilham uma génese comum e são orientadas a sub-problemas específicos. A primeira é orientada a problemas de configuração em tempo de instalação e a segunda lida com a construção de serviços dinamicamente reconfiguráveis.

Existem dois conceitos na arquitectura do *Coyote*: micro-protocolo, que define uma propriedade específica, e de protocolo composto, uma composição de micro-protocolos. A composição de micro-protocolos determina quais as propriedades de um protocolo composto. Os micro-protocolos de uma composição partilham o mesmo espaço de endereçamento e comunicam entre si através de eventos. Cada evento é processado por um gestor (*handler*) definido em tempo de compilação. Um evento pode ser processado por mais do que um gestor. A reconfiguração é conseguida através da definição dos vários gestores para determinados eventos, alternando-se o gestor consoante a necessidade. A necessidade de alternação é sinalizada por um evento específico.

No *Cactus*, para além dos conceitos referidos, subsiste ainda o conceito de serviço. Um serviço é um protocolo composto, em que cada micro-protocolo é responsável por uma propriedade do serviço. Para atingir a reconfiguração dinâmica são construídas várias versões do mesmo serviço, ou seja, vários protocolos compostos distintos. É possível alterar as versões de serviços em tempo de execução.

2.3.2.2 *Appia*

A plataforma *Appia* (Miranda *et al.*, 2001) oferece suporte de concretização e execução de composições de protocolos. Cada protocolo é um módulo que garante um serviço concreto. Uma sessão é uma instância de um protocolo que mantém o estado. Uma pilha de protocolos é denominada uma qualidade de serviço (QdS), dado que define um conjunto de propriedades do qual usufrui o fluxo de mensagens. Um canal é uma instanciação de uma QdS, ou seja, uma pilha de sessões dos protocolos correspondentes. A Figura 2.2 ilustra estes conceitos.

Cada nó de uma rede tem uma ou mais pilhas de protocolos, oferecendo uma ou mais QdS, logo executando um ou mais canais simultaneamente. Além disso,

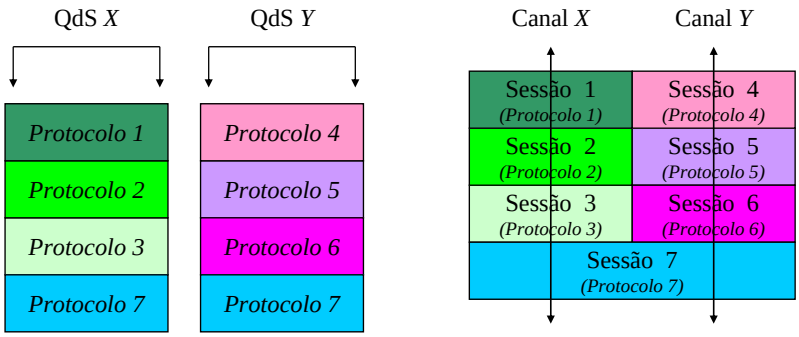

Figura 2.2: Pilhas de protocolos: qualidade de serviço e canais

as sessões podem ser partilhadas por vários canais do mesmo nó. As sessões interagem entre si através de eventos. Cada protocolo deve anunciar quais os eventos que aceita, necessita ou fornece.

O modelo de composição do *Appia* permite ao utilizador definir serviços enriquecidos de comunicação com base na biblioteca de protocolos disponível. Contudo, o modelo não fornece qualquer suporte para a reconfiguração dinâmica das pilhas de protocolos, ou seja, das propriedades dos serviços.

Sumário

Neste capítulo foram definidos os conceitos relacionados com o trabalho abordado nesta dissertação. Foram introduzidos os modelos embebidos e desacoplados e focadas abordagens que utilizam um ou outro. Estas abordagens ora estavam no âmbito do suporte à adaptação, ora na adaptação de sistemas de comunicação. De acordo com os vários aspectos analisados em cada abordagem, propõe-se, no capítulo seguinte, uma arquitectura adequada ao caso estudado neste trabalho.

Capítulo 3

Arquitectura

Neste capítulo são analisadas as necessidades da adaptação de sistemas de comunicação e quais as opções mais adequadas para cada uma. Com base nestas considerações é descrita a arquitectura proposta neste trabalho. São abordados os diferentes componentes e respectivas funções, bem como a dinâmica de interacção entre eles.

3.1 Análise

A construção de um sistema adaptativo a partir de um sistema de comunicação com capacidade de reconfigurar protocolos implica determinar vários aspectos. Estes aspectos são relativos à adaptação, nomeadamente, os estímulos, a lógica de adaptação e quais as alterações a realizar ao sistema. O capítulo anterior permitiu apresentar diferentes abordagens, afinadas para casos específicos, e compreender quais os seus pontos fortes e fracos. No âmbito dos sistemas de comunicação são analisadas de seguida as abordagens nos aspectos relevantes.

Determinar os estímulos da adaptação nos sistemas de comunicação é o primeiro aspecto relevante. A comunicação é afectada pelo estado do sistema, ou seja, por vários factores como as características dos dispositivos e da rede, por exemplo. Esta sensibilidade determina que a adaptação seja realizada em função de alterações nestes factores. Precisamente porque o conjunto destes factores constitui a informação que caracteriza o estado do sistema, é ela que deve estimular a adaptação. A abordagem a seguir na arquitectura deve ser uma ciente

do contexto, sendo o contexto toda a informação relativa aos factores a que a comunicação é sensível. Assim, o contexto engloba preferências do utilizador, os dados a serem transmitidos, as já citadas características dos dispositivos e da rede, entre muitos outros factores.

Uma abordagem ciente do contexto apresenta as vantagens de existir informação detalhada e actualizada dos factores que influenciam a comunicação mas acarreta necessidades. Estas necessidades significam um conjunto de sensores capazes de recolher a informação de contexto e implicam a capacidade de tratar, armazenar e interpretar de acordo com as directivas.

A especificação da reacção aos estímulos derivados do contexto é o segundo aspecto a ter em consideração. Os sistemas de comunicação lidam com diferentes protocolos e informações de contexto. Ao longo do tempo podem ser adicionados novos protocolos ou disponibilizadas outras informações de contexto, mais eficazes. Esta situação implica que seja possível realizar alterações à lógica da adaptação. Desta perspectiva uma abordagem desacoplada permite alterar a lógica de adaptação de forma mais simples do que com uma abordagem embebida. Para além disso, neste cenário, permite ainda reutilizar a lógica de adaptação noutros contextos. Contudo, implica o custo de desenvolver interfaces que permitam o acesso aos elementos a adaptar.

Relativamente às alterações ao comportamento, estas são conseguidas através da reconfiguração dinâmica. A aplicação das reconfigurações, a nível da coordenação, é determinada por estratégias que especificam os vários aspectos intrínsecos à reconfiguração dinâmica de um sistema.

Estas considerações reflectem-se na arquitectura proposta para um sistema de comunicação com capacidade de se reconfigurar, apresentada nas secções que se seguem.

3.2 Modelo da Arquitectura

O modelo da arquitectura proposta segue uma abordagem modular, sendo constituído por diferentes componentes. Os componentes são o gestor da adaptação, o motor da adaptação, o executor da reconfiguração, o monitor de contexto, os

sensores de contexto e os agentes de reconfiguração. Os componentes distinguem-se pela função que desempenham e pela sua abrangência a nível dos nós com que lidam. Na arquitectura os componentes desempenham funções, ou relacionadas com o contexto, ou com a adaptação. Relativamente à função desempenhada, o monitor e os sensores de contexto têm como função tratar da informação de contexto, desde a captura, interpretação e armazenamento à notificação de alterações ao contexto. O gestor da adaptação, o motor da adaptação, o executor da reconfiguração e os agentes de reconfiguração concretizam a lógica da adaptação, desde a determinação das alterações a efectuar à sua aplicação.

Os componentes lidam com vários nós da rede ou apenas com o nó onde estão presentes. Os primeiros são denominados componentes globais: o gestor da adaptação e o monitor de contexto. Os segundos são os componentes locais: os sensores de contexto e os agentes de reconfiguração.

O gestor da adaptação é constituído por vários subcomponentes, a especificação da lógica de adaptação sob a forma de uma política e a concretização das reconfigurações possíveis sob a forma de estratégias. A política é constituída por um conjunto de regras que determinam o comportamento do sistema perante alterações do contexto. O gestor da adaptação é responsável por avaliar a política e aplicar as reconfigurações necessárias. Todos os componentes estão representados na Figura 3.1 que ilustra a arquitectura.

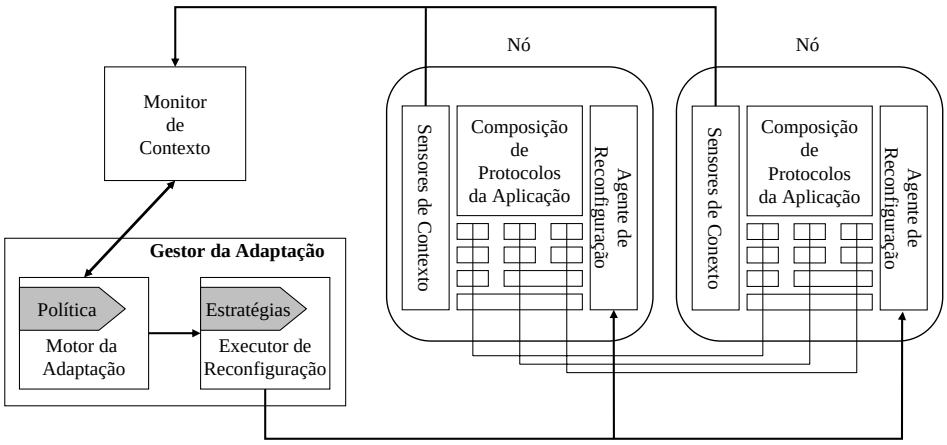

Figura 3.1: Arquitectura proposta

3.3 Componentes

Os elementos da arquitectura são descritos nesta secção. Os sensores de contexto e o monitor de contexto são apresentados na Secção 3.3.1. O gestor da adaptação, motor da adaptação, executor da reconfiguração e os agentes da reconfiguração são descritos na Secção 3.3.2. Finalmente, as políticas de adaptação são introduzidas na Secção 3.3.3.

3.3.1 Contexto

Nesta secção são apresentados os sensores e o monitor de contexto. Estes componentes juntos, oferecem um mecanismo de monitorização do contexto com capacidades de capturar, estruturar e armazenar informação de contexto. Os sensores são responsáveis por capturar informação de contexto relevante e comunicá-la ao monitor. Os sensores não têm conhecimento da informação de contexto nos restantes nós, oferecendo apenas a caracterização do contexto do nó onde estão presentes. O monitor de contexto, em contrapartida, tem uma imagem global do contexto, dado que reúne a informação de contexto dos diferentes nós. Assim, o monitor de contexto, no suporte à adaptação deve oferecer os seguintes serviços:

- Sinalização de alterações significativas ao contexto;
- Satisfação de interrogações (*queries*) sobre o estado do contexto (tipicamente, estes pedidos são projecções directas de condições expressas na política de adaptação).

Na arquitectura, optou-se por concretizar um monitor de contexto centralizado, alimentado por sensores de contexto. A informação capturada é comunicada ao monitor de contexto, que estrutura e interpreta a informação, integrando-a no conjunto de informações de contexto. Desta forma o monitor detecta alterações de contexto relevantes.

O acesso à informação de contexto é realizado através de duas interfaces baseadas em eventos e em interrogações. A primeira interface é concretizada através de eventos publicados pelo monitor e subscritos pelo gestor da adaptação. Desta forma é possível sinalizar as alterações do contexto. A segunda interface permite

ao gestor obter informação de contexto armazenada pelo monitor. Os pedidos de informação são realizados como interrogações ao repositório de informação de contexto. O monitor devolve a informação requisitada.

3.3.2 Adaptação

No controlo da adaptação intervém o gestor da adaptação e os agentes de reconfiguração. O gestor da adaptação, constituído por um motor da adaptação e um executor da reconfiguração, é responsável pela avaliação da política de adaptação e por controlar o processo de reconfiguração. Os agentes de reconfiguração são responsáveis pela aplicação da reconfiguração nos nós a que são locais.

O gestor da adaptação tem como subcomponente o motor da adaptação. O motor é responsável pelo processo de decisão, com base na política de adaptação nele integrada. A avaliação da política baseia-se nos eventos de contexto que lhe são encaminhados pelo gestor da adaptação e em possíveis interrogações que decorrem das regras da política. O motor da adaptação é responsável por comunicar quais as reconfigurações necessárias ao outro subcomponente do gestor: o executor da reconfiguração. O executor é responsável por coordenar a aplicação das reconfigurações de acordo com estratégias nele integradas. O executor comunica com os nós a reconfigurar, enviando a informação de reconfiguração necessária aos agentes de reconfiguração de cada nó.

Os agentes de reconfiguração executam as acções de reconfiguração fornecidas pelo executor. Quaisquer preocupações relativas à manutenção do estado ou coerência são da responsabilidade dos agentes. A reconfiguração é aplicada de acordo com o conhecimento sobre as pilhas a executarem no nó.

3.3.3 Políticas de Adaptação

As políticas de adaptação permitem definir quais as acções de reconfiguração a tomar perante determinadas condições. São constituídas por conjuntos de regras do tipo Evento-Condição-Acção (ECA) permitindo definir quais os elementos que as activam, as condições que se aplicam e quais as acções de reconfiguração a executar. A política é avaliada pelo gestor da adaptação sempre que este recebe um evento. As condições são verificadas através de pedidos de informação ao

monitor de contexto. Desta forma são determinadas as acções de reconfiguração. As políticas podem ainda ser reutilizadas noutros contextos distintos.

3.4 Dinâmica

A dinâmica da arquitectura é o conjunto das interacções entre os vários componentes da da arquitectura. Estas interacções ocorrem entre os componentes de contexto (entre os sensores e o monitor) e entre os componentes adaptação (entre o gestor e os agentes de reconfiguração). Existe ainda interacção entre o monitor de contexto e o gestor da adaptação. A Figura 3.2 ilustra o conjunto de interacções da arquitectura.

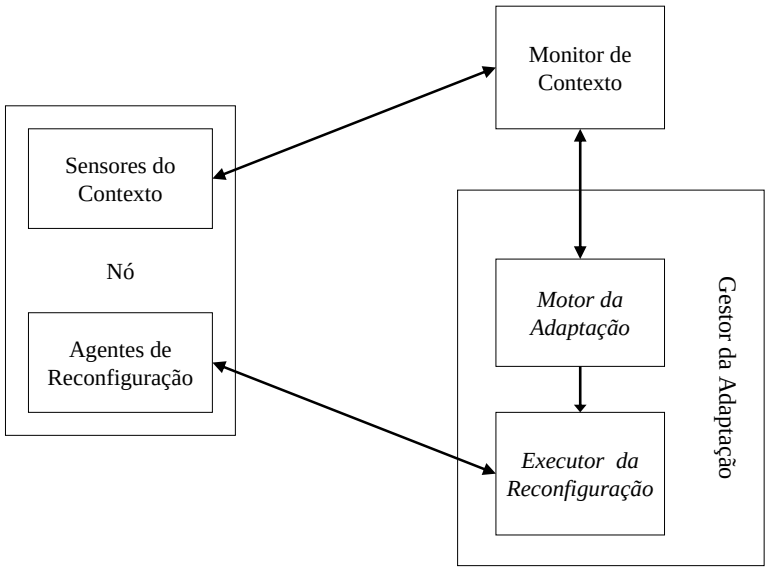

Figura 3.2: Dinâmica global da arquitectura.

A dinâmica inicia-se com a captura de informação pelos sensores. Os sensores comunicam a informação ao monitor de contexto. Esta comunicação pode acontecer nos dois sentidos. O monitor é responsável por enviar eventos a sinalizar alterações de contexto. Estes eventos são recebidos pelo motor da adaptação. O motor de adaptação pode requisitar informação adicional ao monitor de contexto, através de interrogações. O motor comunica as reconfigurações resultantes

da avaliação da política ao executor da reconfiguração. O executor da reconfiguração troca a informação necessária com os agentes de reconfiguração e estes confirmam a conclusão da reconfiguração ao executor.

Sumário

Neste capítulo foi proposta a arquitectura genérica para o suporte à construção de sistemas adaptativos, de acordo com as conclusões retiradas do capítulo anterior. Foram descritos os diversos componentes da arquitectura, bem como a sua interacção, e introduzida a especificação da política de adaptação utilizada na arquitectura. Uma descrição pormenorizada da política de adaptação é realizada no capítulo seguinte

Capítulo 4

Políticas de Adaptação

A arquitectura apresentada no Capítulo 3 segue uma abordagem orientada a políticas, explícitas, promovendo uma separação clara entre os aspectos funcionais e não funcionais. Esta aproximação oferece várias vantagens. Em primeiro lugar, a definição das políticas não exige a compreensão dos pormenores da concretização, nomeadamente dos objectos a adaptar. Assim, esta tarefa pode ser realizada por intervenientes que não tenham conhecimento da concretização e facilita a definição da política, ao excluir os pormenores irrelevantes de concretização. Em segundo lugar, simplifica a análise e manutenção do sistema, uma vez que os aspectos adaptativos e funcionais podem ser estudados e optimizados separadamente. Finalmente, promove a reutilização da lógica de adaptação noutros contextos distintos.

As vantagens descritas podem ser materializadas num suporte rico e flexível à adaptação. Neste capítulo apresenta-se a concretização deste suporte, com ênfase na linguagem de especificação de políticas de adaptação, sem deixar de referir os modelos de suporte necessário para a sua interpretação. A linguagem segue uma estrutura genérica adequada às necessidades adaptativas em abordagens cientes do contexto. No entanto, as acções de reconfiguração são direccionadas para um domínio específico: a adaptação de sistemas de comunicação desenvolvidos como uma composição vertical (em pilha) de protocolos.

A adaptação orientada a políticas pressupõe modelos de contexto, a especificação dos alvos da adaptação e estratégias para aplicar a reconfiguração. Os modelos de contexto definem de forma abstracta toda a informação de contexto

necessária para avaliar o estado do ambiente, bem como a interface disponibilizada para aceder à informação. A especificação dos alvos da adaptação permite definir os elementos que podem ser referidos para reconfiguração. Por fim, as estratégias determinam de que maneira as acções de reconfiguração da política são aplicadas.

4.1 Modelos de Contexto

Na arquitectura proposta, a informação de contexto é um aspecto fundamental na adaptação, reflectindo o estado do ambiente externo e, eventualmente, de partes do estado do sistema. A adaptação depende do estado do contexto, pelo que a definição da política de adaptação é condicionada pela informação disponível. O modelo de contexto define qual a informação capturada pelos sensores e a sua interpretação, ou seja, a informação relevante e necessária. O modelo também descreve as interfaces de acesso à informação.

O modelo de contexto é a abstracção do contexto relevante. Este modelo é crucial para a definição da política, descrevendo qual a informação disponível e de que forma pode ser acedida. A Figura 4.1 descreve o papel do modelo de contexto relativamente ao monitor de contexto e gestor da adaptação.

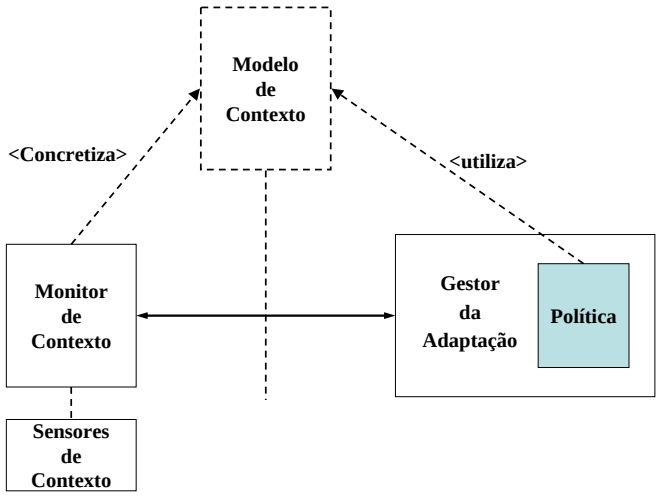

Figura 4.1: Papel do Modelo de Contexto

O modelo de contexto condiciona o funcionamento dos vários componentes da arquitectura: sensores, monitor de contexto e gestor da adaptação. Do ponto de vista dos sensores, o modelo define os aspectos do ambiente a capturar. Estes aspectos são informação de contexto que pode ser acedida directamente, sem necessidade de interpretação. A interpretação é relegada para o monitor de contexto, pelo que o modelo descreve quais as interpretações da informação capturada e as interfaces de acesso à mesma. Para o gestor da adaptação, o modelo de contexto corresponde a um compromisso entre as necessidades de informação de contexto e o custo de captura e monitorização. Assim, a política baseia-se nos eventos oferecidos no modelo de contexto e nas interrogações (*queries*) disponíveis para aceder à informação.

Neste trabalho, o modelo de contexto adoptado seguiu a abordagem proposta por Lopes & Fiadeiro (2005). Nesta abordagem, o modelo é definidos através de informações observáveis e derivadas. No entanto, dado que a abordagem não considera a notificação de alterações de contexto, o modelo foi estendido com a capacidade de definir eventos. As informações observáveis são capturadas pelos sensores ou, então, são informação estática de cada nó. As informações derivadas são interpretadas a partir de informação observável. Tanto a informação observável como a informação derivada está associada a um tipo de contexto. O acesso à informação é realizado através de interrogações. As interrogações possíveis estão definidas no modelo de contexto. O exemplo seguinte demonstra a definição de informações observáveis e derivadas, bem como os seus modos de acesso:

```
contextType Network
        sensed observable isLaptop: nodeId -> bool
        sensed observable isWired: nodeId -> bool
        derived observable isStableNode: nodeId -> bool
          isStableNode(nodeId)=(!isLaptop(nodeId) && isWired(NodeId))
```

No exemplo é definido o tipo de contexto (*contextType*) *Network* com duas informações observáveis (*sensed observable*) e uma derivada (*derived observable*). Nas informações observáveis, tanto *isLaptop* como *isWired* podem ser acedidas através de uma interrogação contendo a identificação do nó (*nodeId*). A resposta à interrogação é dada através de um booleano. A informação interpretada

isStableNode é definida através das duas observáveis já descritas. O acesso à informação é através de um *nodeId*, que permite verificar se a observável *isLaptop* toma um valor falso e a *isWired* toma um valor verdadeiro.

A manipulação da informação de contexto é realizada de forma directa e baseia-se em tipos e operações internas. Relativamente aos tipos, consideram-se os tipos primitivos (*int, bool, double, ...*) e tipos fundamentais como o *nodeId*, que representa a identificação de um nó. Sem este tipo seria impossível identificar os nós onde ocorrem alterações de contexto ou quais os nós alvo da adaptação. As operações possíveis são as associadas aos tipos primitivos, como as operações relacionais e lógicas.

Relativamente aos eventos, estes sinalizam alterações de contexto relevantes, podendo incluir informação relativa à transição de um estado para outro. Esta transição pode ser um aspecto de acesso directo (p.e., valor da largura de banda) ou um conjunto de informações em situações bem definidas (p.e., caracterização da comunicação com base em taxas de erro, latência e débito). Os eventos são definidos no modelo de contexto. Para a sua declaração é necessário o nome do evento, a informação que disponibiliza e o acesso à mesma. O exemplo seguinte demonstra a declaração dos eventos *NewConnectionEvent* e *ErrorStartEvent*:

```
event NewConnectionEvent
        parameter nodeId : nodeId
        parameter wired : bool

event ErrorEvent
        parameter nodeId : nodeId
        parameter type: String
```

No exemplo são declarados dois eventos através da palavra reservada *event* e, para cada evento, parâmetros através da palavra *parameter*. É também indicado o tipo de cada parâmetro. Em ambos os eventos existe o parâmetro *nodeId* do tipo *nodeId* que representam a identificação do nó onde ocorreu a alteração assinalada. No primeiro evento existe ainda um parâmetro *wired* do tipo *bool* e no segundo evento, o parâmetro *type* do tipo *String*.

A concretização e utilização do modelo de contexto é abordada nas secções que se seguem, desde a captura de informação pelos sensores, à recolha e interpretação da informação pelo monitor de contexto e até à definição de políticas.

4.2 Alvos da Adaptação

Na secção anterior ficou presente que os eventos que despoletam as regras, assim como as interrogações que descrevem as restrições à aplicação, podem ser nomeados utilizando o modelo de contexto. Nesta secção descrevem-se os elementos alvo da adaptação, bem como a sua referência nas regras da política de adaptação.

Nesta dissertação, as políticas de adaptação são orientadas a sistemas de comunicação que seguem modelos de composição semelhantes ao da plataforma *Appia*. Neste modelo existem diversos elementos que podem ser referidos na especificação da política. Estes elementos são descritos de seguida.

4.2.1 Modelo de Composição da Plataforma *Appia*

A plataforma *Appia*, já descrita na Secção 2.3.2.2, utiliza um modelo de composição de protocolos para assegurar a comunicação numa rede com vários nós. Recordando, cada nó executa uma ou mais pilhas de protocolos que lhe permitem comunicar com outros nós. Cada pilha oferece uma qualidade de serviço caracterizada por uma composição vertical de protocolos, em que cada protocolo é responsável por imprimir determinadas propriedades ao fluxo de mensagens que atravessa a pilha. Designa-se a instância de um protocolo por *sessão*. Desta forma, uma instância de uma qualidade de serviço, designada por *canal*, é constituída por uma pilha de sessões. A Figura 4.2 ilustra vários nós de uma rede, bem como os canais activos em cada um.

Dado o modelo de composição da plataforma *Appia*, os seguintes elementos podem ser referidos numa política: protocolos, canais, sessões e nós da rede, onde os outros elementos residem. Os protocolos e os canais exigem uma tipificação e, também, a organização em hierarquias. Os restantes elementos não necessitam de um tratamento semelhante. A forma como os elementos são referidos é abordada nas secções que se seguem.

4.2.2 Protocolos

Na definição de protocolos, sempre que é necessário nomear um protocolo, recorre-se a uma hierarquia de tipos de protocolos. Em geral, uma hierarquia de tipos

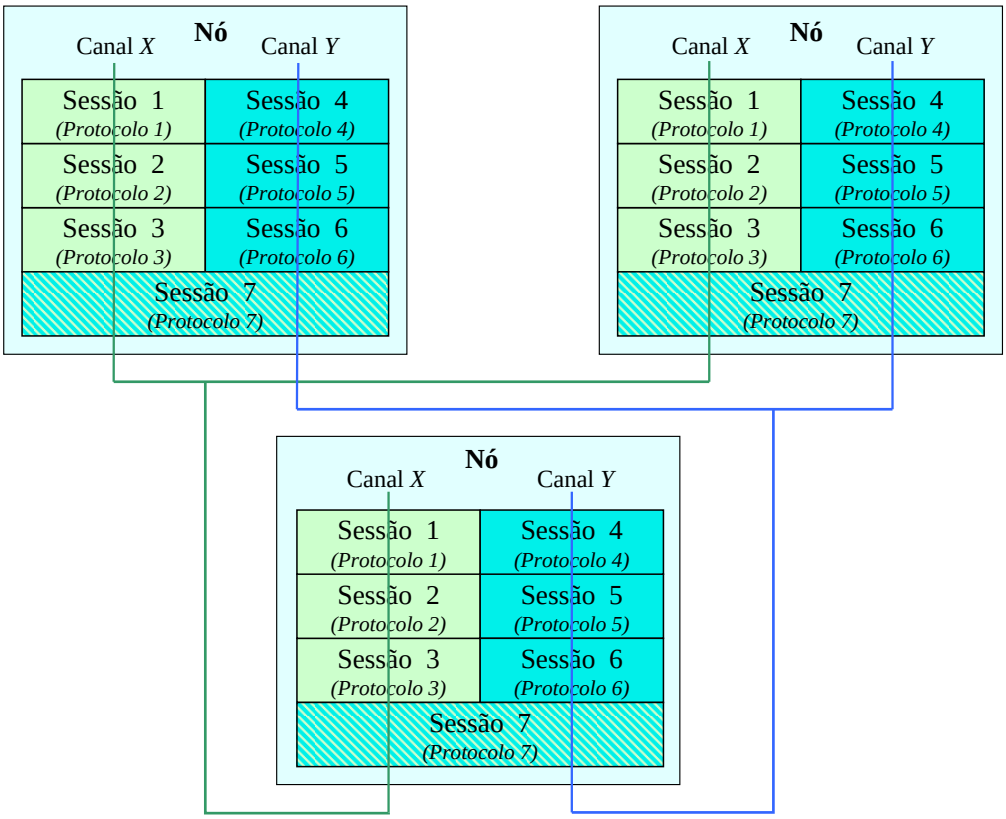

Figura 4.2: Canais em cada nó

representa a organização de um conjunto de elementos de acordo com propriedades relevantes para um determinado ponto de vista. Por outras palavras, uma hierarquia permite agrupar diferentes elementos, que partilham características relevantes para o contexto da aplicação, sobre uma mesma denominação.

A utilização de hierarquias de tipos apresenta diversas vantagens, que se prendem com a configuração da pilha e com o grau de abstracção da política. A configuração das pilhas é dinâmica, evoluindo à medida que várias adaptações vão sendo efectuadas. Mesmo conhecendo a configuração inicial, a configuração actual depende sempre da sequência de acções de reconfiguração sofridas pela pilha. Desta forma, o conhecimento da configuração exacta da pilha é difícil. A definição das regras da política de adaptação lida com este problema. A utilização de uma hierarquia de tipos permite definir um conjunto de alvos que partilham

características específicas, ao invés de um alvo bem determinado. Assim, não é necessário conhecer qual o protocolo alvo na pilha mas sim qual o serviço que o protocolo alvo oferece. A definição dos tipos de protocolos, bem como a hierarquia, é abordada de seguida.

4.2.2.1 Hierarquia de Tipos de Protocolos

A definição de uma hierarquia passa por um conjunto de tipos concretos e abstractos, bem como de relações de subtipos. Numa hierarquia existe sempre um tipo fixo, que é a raíz da hierarquia. Um tipo é concreto quando se refere a um protocolo específico, ao passo que é considerado abstracto quando define múltiplos protocolos concretos ou abstractos. Um tipo abstracto admite um conjunto variado de protocolos como subtipos. As relações entre tipo e subtipo podem ser directas ou indirectas. No primeiro caso, o relacionamento é directo se for o tipo imediatamente abaixo ou acima, ou seja, de subtipo directo. O segundo caso refere-se às restantes situações.

Na hierarquia de tipos de protocolos, os tipos concretos representam protocolos em particular e os abstractos representam tipicamente qualidades de protocolos ou serviços que oferecem. O tipo abstracto *Protocol* é a raíz da hierarquia em árvore, portanto todos os tipos de protocolos são subtipos deste. Todas as folhas da árvore representam tipos de protocolos concretos e os restante nós ora tipos concretos, ora abstractos. Consideram-se as seguintes expressões para denotar conjuntos de tipos de protocolos abstractos ou concretos:

```
ProtocolTypeExp := ◇ProtocolType | ProtocolType |
                   ProtocolTypeExp and ProtocolTypeExp |
                   ProtocolTypeExp or ProtocolTypeExp
```

A semântica desta expressão, *ProtocolTypeExp*, é a seguinte:

- $\diamondsuit$*ProtocolType* denota um conjunto singular apenas com o protocolo concreto *ProtocolType*;
- *ProtocolType* denota o conjunto de subprotocolos concretos relacionados directa e indirectamente (verticalmente) com o tipo *ProtocolType*, incluindo o próprio tipo;

- *ProtocolTypeExp* **and** *ProtocolTypeExp* denota o conjunto de protocolos resultante da relação de intersecção de expressões *ProtocolTypeExp*;

- *ProtocolTypeExp* **or** *ProtocolTypeExp* denota o conjunto de protocolos relacionados com uma das expressões *ProtocolTypeExp* presentes, logo a união das várias relações directas e indirectas.

A principal consideração na construção da hierarquia de protocolos é organizar o conjunto de protocolos considerado relevante no contexto da aplicação, de acordo com as funcionalidade que cada protocolo oferece. Esta aproximação permite referir os protocolos alvos da adaptação de acordo com as necessidades adaptativas.

Na Figura 4.3 apresenta-se um exemplo de uma hierarquia de tipos de protocolos. Neste exemplo os tipos abstractos estão representados a itálico. O tipo *Protocol* é a raíz da árvore e dela partem os subtipos *TransportProtocol* e *FragmentationProtocol*. Os restantes tipos representados na hierarquia são folhas da árvore e, como tal, representam os protocolos concretos: *UDPTransportProtocol* e *TCPTransportProtocol*. Este último protocolo é subtipo de dois tipos abstractos: *TransportProtocol* e *FragmentationProtocol*. Uma expressão *TypeExp1 and TypeExp2*, em que *TypeExp* são os tipos abstractos referidos, resultaria não num conjunto vazio mas num conjunto com um elemento: *TCPTransportProtocol*. O mesmo exemplo, mas utilizando a partícula *or* (união), resultaria nos dois tipos de protocolos concretos que são folhas da árvore.

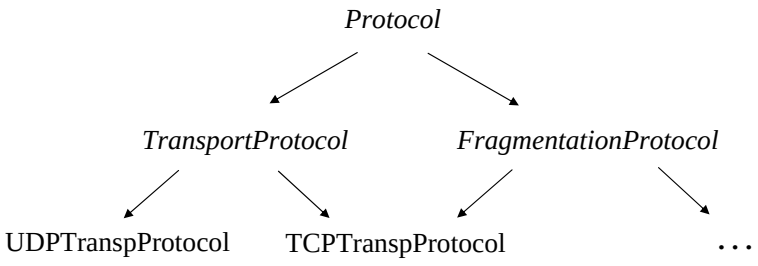

Figura 4.3: Hierarquia de tipos de protocolos

4.2.3 Canais

Para referir um canal recorre-se a uma hierarquia de tipos de canais, tal como no caso dos protocolos. Os canais são tipificados e organizados por uma hierarquia descrita de seguida.

4.2.3.1 Hierarquia de Tipos de Canais

A hierarquia de tipos de canais considerada é semelhante à de tipos de protocolos. A estrutura é igualmente em árvore, em que todos os nós folha representam tipos de canais concretos e os restantes nós tipos abstractos ou concretos. A raíz da árvore é o tipo abstracto *Channel*, do qual descendem todos os subtipos de canais. Consideram-se as seguintes expressões para denotar conjuntos de tipos de canais abstractos ou concretos:

```
ChannelTypeExp := ◇ChannelType | ChannelType |
                  ChannelTypeExp and ChannelTypeExp |
                  ChannelTypeExp or ChannelTypeExp
```

A semântica da *ChannelTypeExp*, semelhante à dos tipos de protocolos, é a seguinte:

- $\diamondsuit$*ChannelType* denota um conjunto singular apenas com o canal concreto *ChannelType*;
- *ChannelType* denota o conjunto de canais concretos relacionados directa e indirectamente com o tipo *ChannelType*;
- *ChannelTypeExp* **and** *ChannelTypeExp* denota o conjunto de canais resultante da relação de intersecção de expressões *ChannelTypeExp*;
- *ChannelTypeExp* **or** *ChannelTypeExp* denota o conjunto de canais relacionados com uma das expressões *ChannelTypeExp* presentes, logo a união das várias relações directas e indirectas.

A hierarquia de tipos de canais permite referir os canais alvos da adaptação de acordo com as necessidades adaptativas. Para organizar os canais numa hierarquia, existem várias abordagens possíveis. Uma abordagem seria a organização

dos canais em função da configuração dos mesmos. No entanto, a mutabilidade da configuração e o facto da mesma configuração poder servir fins distintos revelam ambiguidade do critério. Em contrapartida, preferir a organização dos canais em função do fim a que se destinam, ou seja, com base na função que desempenham na aplicação resolver os problemas referidos. Da perspectiva da declaração da política de adaptação, esta última abordagem é a mais vantajosa, dado que os canais alvos são escolhidos de acordo com a função que desempenham no contexto da aplicação.

Na Figura 4.4 apresenta-se um exemplo de organização dos canais pela função que desempenham a nível da aplicação. Neste exemplo os tipos abstractos estão representados a itálico e são as folhas da árvore: *AudioChannel* e o *DataChannel*. O tipo abstracto *AudioChannel* tem dois subtipos concretos: os canais *ControlChannel* e *VoiceChannel*.

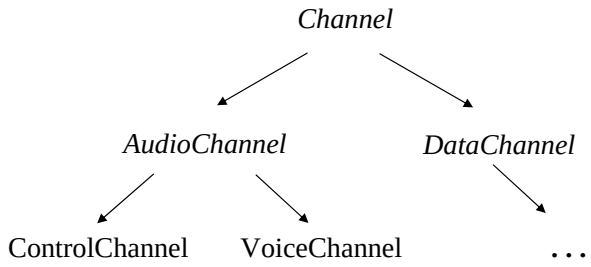

Figura 4.4: Hierarquia de canais

4.2.4 Sessões

As sessões, ao contrário dos anteriores elementos, não são referidas explicitamente na política de adaptação. Tal deve-se ao facto das sessões apenas ficarem decididas durante a execução. No entanto, a referência implícita a sessões é possível, através de indicações de tipo de canal e de tipo de protocolo, o que será esclarecido mais à frente.

4.2.5 Nós da Rede

Todos os nós da rede têm uma identificação única sobre a forma de um tipo *nodeId*, abordado na Secção 4.1. Para referir nós na política de adaptação, utiliza-se um conjunto de *nodeId*. Este conjunto é representado entre chavetas, {}, determinando o conjunto de nós. Este conjunto também pode ser o resultado de uma informação observável (*observable*), definida no modelo de contexto, que devolve um conjunto de identificações de nós.

4.3 Linguagem de Definição de Políticas

Nesta secção é abordada a linguagem de definição de políticas de adaptação para pilhas de protocolos. São abordados os vários aspectos da linguagem, as acções de reconfiguração e as estratégias de reconfiguração presentes na arquitectura. Os vários aspectos da linguagem referem-se à estrutura, primitivas e hierarquia de tipos necessária. As acções determinam a reconfiguração da pilha protocolar, sendo abordado o âmbitos das acções, bem como a situação a que cada uma das acções é orientada e qual o seu efeito específico. Para ilustrar cada primitiva ou acção de reconfiguração são fornecidos exemplos genéricos do ponto de vista da optimização do desempenho. Por fim, as estratégias concretizam as acções de reconfiguração das pilhas de protocolos, sendo descrita a aproximação para cada acção.

4.3.1 Estrutura das Políticas

A política de adaptação é um conjunto de regras, em que cada regra define quando e como adaptar o quê. Com essa consideração, as regras de uma política seguem um formato Evento-Condição-Acção (ECA) proposto por McCarthy & Dayal (1989). Através deste formato é possível especificar quais as alterações que despoletam a adaptação, quais as condições e factores que têm de se verificar e qual adaptação a realizar. As alterações do estado relevantes são descritas na regra através de eventos. Os *eventos* permitem definir a alteração ou o conjunto de alterações que exigem uma modificação da pilha protocolar. As *condições* são constrangimentos aplicados a determinados aspectos, que não estão directamente

relacionadas com as alterações assinaladas pelos eventos. Por fim, a adaptação é uma ou várias *acções* de reconfiguração aplicadas à pilha de protocolos num nó ou conjunto de nós.

A avaliação de uma regra implica três fases distintas. A primeira fase consiste em despoletar a regra, verificando-se os eventos definidos; a segunda fase consiste em activar a regra, satisfazendo-se as condições aplicáveis; e a terceira fase consiste no efeito da regra, sendo conhecido o conjunto de passos de configuração que devem ser aplicados. Só após a primeira e a segunda fases é que uma regra se considera activa, servindo a terceira fase para determinar quais as modificações necessárias e os alvos das mesmas.

Nas políticas de adaptação, as acções determinam a reconfiguração das pilhas protocolares. A aplicação das acções pode ser global ou selectiva. Neste último caso apenas um subconjunto é afectado pela acção. Para tal, é associado um âmbito a cada acção. O âmbito limita o raio de acção a um conjunto de nós, protocolos ou mesmo os canais que servem um fim específico. Assim, existem âmbitos distintos para circunscrever o raio da acção em diferentes elementos: nós, protocolos e canais de comunicação. No caso do primeiro elemento, o âmbito descreve quais os nós afectados através de um conjunto de identificações *nodeId*. Este conjunto pode ser definido recorrendo-se a características dos nós (nó móvel e nó fixo, por exemplo) ou outra informação (tipo de conexão, taxa de erros, entre outras), desde que definida no modelo de contexto. No segundo e terceiro elementos o âmbito é definido através de hierarquias de tipos de protocolos e de canais, já descritas nas Secções 4.2.2.1 e 4.2.3.1.

4.3.2 Âmbitos

A aplicação de uma acção de reconfiguração pode ser global ou selectiva. O âmbito determina qual o conjunto de elementos afectados pela acção. A omissão do âmbito da acção determina que a aplicação é global, ou seja, todos os nós, protocolos e canais são afectados. O âmbito de uma acção tem três primitivas que definem quais os nós, protocolos e canais afectados. O âmbito topológico define quais os nós que são afectados, o âmbito de protocolos determina qual o conjunto de protocolos alvo e o âmbito de canais contempla o tipo de canais que sofrem a

acção. Os vários âmbitos são abordados mais detalhadamente nas secções que se seguem.

4.3.2.1 Topológico

Na arquitectura proposta neste trabalho considera-se que cada nó tem um identificador único do tipo *nodeId*. O âmbito topológico é definido através de um conjunto de identificadores *nodeId*. Este conjunto pode ser definido de duas formas: por extensão ou por intensão. A primeira passa por reunir as identificações dos nós afectados entre chavetas: $\{nodeId1, nodeId2, nodeId3\}$. Esta abordagem apenas é possível se os eventos fornecerem informação sobre a identificação dos nós. Um exemplo de âmbito topológico por extensão é a definição de uma regra que, perante a diminuição da largura de banda num nó reconfigura o protocolo de transporte, aumentando o tempo de *timeout*. Relativamente à segunda forma, por intensão, esta consiste em definir uma ou várias características dos nós alvo. São seleccionadas todas as identificações dos nós que reúnam as características definidas, construindo um conjunto. Um exemplo deste âmbito é definir todos os nós que sejam dispositivos móveis, acedendo à informação de contexto necessária.

4.3.2.2 Protocolos

Uma sessão é uma instância de um protocolo que está presente na configuração de um ou mais canais. Num nó podem existir várias instâncias de um mesmo protocolo em diferentes canais ou uma instância ser partilhada por mais que um canal. Esta situação implica que o âmbito de protocolos seja flexível o suficiente para lidar com estas situações e dar resposta às necessidades da política de adaptação. Assim, o âmbito de protocolos lida com as sessões afectadas pela acção de reconfiguração.

O âmbito de protocolos, para definir as sessões afectadas, necessita de diferentes informações. A principal informação é o tipo de protocolo alvo. Dado que uma sessão pode pertencer a mais do que um canal do nó, é necessária informação sobre o tipo de canal que é afectado pela acção. Para definir estas informações são utilizadas expressões que descrevem tipos de protocolos e canais, logo podem ser tipos abstractos, existindo mais do que um protocolo do mesmo tipo num

canal. Nestes casos recorre-se a uma localização relativa para distinguir as sessões que devem ser afectadas das restantes. A localização relativa consiste em palavras reservadas e um tipo de protocolo de referência. A definição do âmbito de protocolos, como descrito, é realizada através de uma expressão *SessionExp*:

```
SessionExp :=
  ProtocolTypeExp |
  ChannelTypeExp |
  SessionExp below ProtocolTypeExp |
  SessionExp above ProtocolTypeExp |
  SessionExp and SessionExp |
  SessionExp or SessionExp
```

A semântica desta expressão é a seguinte:

- *ProtocolTypeExp* denota sessões de protocolos do tipo em questão;
- *ChannelTypeExp*denota sessões pertencentes a canais do tipo em questão;
- *SessionExp*:
  - *below* denota a localização abaixo de um protocolo de referência *ProtocolTypeExp*;
  - *above* denota a localização acima de um protocolo de referência *ProtocolTypeExp*;
  - *and* denota a intersecção de sessões *SessionExp*;
  - *or* denota a união de sessões *SessionExp*.

Tanto a intersecção de sessões *SessionExp*, como a união, permitem definir o âmbito de protocolos.

4.3.2.3 Canais

A necessidade de um âmbito de canais deve-se a situações em que existem sessões partilhadas entre canais diferentes. O âmbito de protocolos permite identificar qual a sessão alvo. Assim, sempre que a sessão é partilhada por vários canais, a acção de reconfiguração afecta todos. Com esta abordagem, o efeito desejável

num canal pode não o ser para os restantes. Nestas situações, o âmbito de canais permite distinguir entre afectar todos os canais ou apenas alguns.

A definição do âmbito de canais consiste apenas numa expressão *ChannelTypeExp*. Esta expressão especifica qual o tipo de canais afectados pela acção de reconfiguração. Nas situações, com sessões partilhadas, em que o âmbito de canais é omitido, a adaptação afecta todos os canais que contêm a sessão partilhada.

4.3.3 Primitivas

Uma política consiste num conjunto de regras que definem quais as condições em que determinadas acções de reconfiguração têm lugar. A declaração das regras da política recorre a uma linguagem específica. Esta linguagem assenta num conjunto de primitivas e numa sintaxe bem definida. A sintaxe respeita o formato ECA das regras da política e reflecte a necessidade de aplicar um âmbito às acções de reconfiguração. As primitivas são *When*, *With*, *Do*, *Where*, *For* e *Apply*. As primeiras três referem-se ao formato evento, condição e acção, enquanto que as restantes correspondem a cada um dos âmbitos de acção. A sintaxe para a definição das regras da política é a seguinte:

```
WHEN triggerCondition
 [WITH stateCondition]
DO
  {reconfigurationAction
  [WHERE topologicalScope]
  [FOR protocolScope][APPLY channelScope]}+
```

4.3.3.1 *When*

Na declaração de uma regra da política, a primitiva *When* permite definir quais os eventos que despoletam a regra em causa. Os eventos assinalam alterações relevantes no contexto e, em alguns casos, contêm informação relacionada. A declaração da primitiva *When* é obrigatória em todas as regras. Esta primitiva envolve uma expressão *triggerCondition*, que descreve qual o ou os eventos que despoletam a regra, bem como qualquer condição que se aplique à informação contida nos eventos. Dado que nem todas as ocorrências de um evento são significativas para a reconfiguração, a utilização de condições permite distinguir quais

as instâncias de eventos que realmente são relevantes para a regra em questão. A sintaxe da expressão *triggerCondition* é a seguinte:

```
event: condition {or event: condition}+
```

Cada módulo da expressão é constituído por um par *event-condition*. O *event* é o nome do evento e *condition* a condição aplicada à informação do evento. A declaração do primeiro elemento é obrigatória e do segundo é facultativa. Diferentes módulos podem ser combinados na declaração da primitiva através das operações lógicas *e* e *ou*. A primeira combinação é conseguida sem necessidade de partícula de ligação, ao passo que a segunda ligação exige a presença da partícula *or*. Para ilustrar a declaração da primitiva *When* apresenta-se o seguinte exemplo:

```
WHEN BandwidthEvent: BandwidthEvent.value < MIN
     or JoinNodeEvent
```

Neste exemplo, os eventos *BandwidthEvent* ou *JoinNodeEvent* despoletam a regra em causa. No caso do primeiro evento, aplicam-se as restrições à informação contida. A condição imposta declara que o valor de *value* tem de ser inferior a MIN. Só cumprindo esta restrição a regra é despoletada.

4.3.3.2 *With*

A segunda primitiva da sintaxe apresentada declara restrições ao estado do sistema ou do ambiente de execução. Estas condições não se referem à informação contida em eventos, logo resulta numa interrogação ao monitor de contexto, pedindo informação específica. Quando as condições declaradas na primitiva *With* são respeitadas, a regra é activada. No entanto, dado que a declaração da primitiva *With* é facultativa, nos casos de omissão da primitiva, a declaração da primitiva *When* determina o despoletar e activar da regra em causa.

As condições são declaradas através de operações lógicas, aritméticas ou relacionais entre características observáveis definidas no modelo de contexto. As características são definidas seguindo a sintaxe *contextType.observable*. A especificação de condições é ilustrada pelo exemplo seguinte:

```
WITH Network.numberOfNodes >10 &&
     Network.numberOfWiredNodes >3
```

Neste exemplo, são declaradas duas condições. Ambas as condições restringem características do *contextType Network*. A imposição à característica observável *numberOfNodes* (número de nós da rede) é que esta seja superior a dez, ao passo que a característica *numberOfWiredNodes* (número de nós fixos da rede) tem de ser superior a três.

4.3.3.3 *Do*

Após o despoletar e activar de uma regra, a declaração da primitiva *Do* é obrigatória e determina quais as acções de reconfiguração a concretizar. A declaração de uma acção de reconfiguração consiste em identificar a acção, fornecer os parâmetros eventualmente necessários e qual o âmbito da acção (o conjunto de alvos). No caso de várias acções, é definido um âmbito para cada uma, mesmo nas situações em que é igual. As declarações de acções são separadas pela partícula *and*.

Nos casos em que são declaradas várias acções, este conjunto pode ser compreendido de duas perspectivas possíveis. Uma perspectiva passa por considerar que as acções são executadas pela ordem como foram definidas. A outra perspectiva é considerar que o conjunto de acções de reconfiguração é uma acção composta. A nível da política não é dada nenhuma garantia, pelo que a execução das acções de reconfiguração não é determinista.

O exemplo seguinte ilustra a declaração de acções sem âmbito. As declarações de acções com âmbito serão exemplificadas na próxima secção.

```
DO reconfigurationAction1 and
       reconfigurationAction2
```

4.3.3.4 *Where*, *For* e *Apply*

Estas três primitivas declaram o âmbito de uma acção de reconfiguração. A sua declaração é facultativa. Se o âmbito não for explicitamente declarado, a acção, por omissão, afectará todos os nós, protocolos e canais. A primitiva *Where* define o âmbito topológico através de um conjunto de identificações dos nós. As identificações são descritas através de *nodeId*. A primitiva *For* define o âmbito de protocolos, através de uma expressão *SessionExp*. A primitiva *Apply* define o

âmbito de canais, através de uma expressão *ChannelTypeExp*. Um exemplo da declaração do âmbito de uma acção é apresentado de seguida:

```
WHEN ErrorEvent
DO reconfigurationAction
   WHERE ErrorEvent.NodeId
   FOR DebugProtocol above TransportProtocol
   APPLY DataChannel
```

Neste exemplo é apresentada a declaração de uma regra activada apenas pelo evento *ErrorEvent*, dado que que a declaração da primitiva *With* é omissa. A declaração da primitiva *Do* determina a acção a aplicar: *reconfigurationAction*. O âmbito desta acção é constituído pela declaração das três primitivas de âmbito. A primitiva *Where* define que apenas o nó onde ocorreu o evento é afectado pela acção em causa. A primitiva *For* declara que as sessões afectadas são do tipo *DebugProtocol* que, na pilha de protocolos, estejam numa posição acima (*above*) de uma sessão do tipo *TransportProtocol*. No caso de existirem sessões partilhadas, a primitiva *Apply* declara que apenas as sessões do canal do tipo *DataChannel* são afectadas pela acção de reconfiguração.

4.3.4 Acções de Reconfiguração

As acções de reconfiguração são, tipicamente, dependentes do domínio de aplicação da política. Nesta dissertação, estas acções são expressas como uma ou várias operações sobre a pilha de protocolos.

As acções distinguem-se em dois tipos, de acordo com o seu efeito. Estas ou incidem ao nível de um protocolo específico ou, então, sobre a configuração da pilha. No primeiro caso, as acções permitem afinar os protocolos, alterando os valores de parâmetros de configuração específicos. No segundo caso, as acções de reconfiguração alteram a qualidade de serviço, através de operações de introdução, remoção e substituição de protocolos, bem como a completa substituição da pilha.

As acções de reconfiguração apresentadas nas secções que se seguem são fruto de uma análise das necessidades dos utilizadores na plataforma de composição e execução de protocolos *Appia*. As acções reflectem as necessidades sentidas em diversas aplicações como jogos distribuídos em tempo real (Rodrigues *et al.*, 2001), replicação de bases de dados (Rodrigues *et al.*, 2002), ambiente multi-utilizador orientados a objectos (Teixeira *et al.*, 2002), aplicações colaborativas em

redes móveis (Mocito *et al.*, 2005), entre outras. Para cada acção é apresentada a sintaxe e um exemplo ilustrativo.

4.3.4.1 *setValue*

A acção de reconfiguração *setValue* tem por objectivo a afinação de parâmetros de protocolos, a nível individual. A reconfiguração dinâmica do valor dos parâmetros permite optimizar o desempenho do protocolo visado. Esta acção aceita os três âmbitos distintos e necessita de informação relativa ao parâmetro a afinar e qual o novo valor. A sintaxe é a seguinte:

$$setValue(\text{parameter}, \text{newValue})$$

O elemento *parameter* identifica qual o parâmetro a alterar para o novo valor *newValue*. O protocolo ou tipo de protocolo afectados pela acção é declarado através do âmbito de protocolos e de canais. Os nós afectados são definidos pelo âmbito topológico. Se o âmbito resultante da conjunção dos vários âmbitos resultar num conjunto vazio ou os alvos não pertencerem à pilha, a acção não tem efeito.

Quando um alvo de uma acção de reconfiguração é uma sessão partilhada, a aplicação pode ter dois efeitos distintos. O primeiro ocorre quando o âmbito de canais é omisso. O segundo ocorre quando este está presente. Os efeitos são os seguintes:

- No primeiro caso, pressupõe-se que o âmbito de canais é omitido ou que o âmbito definido é igual ao conjunto de canais que partilham a sessão. Este cenário não define um canal ou canais alvo específicos, pelo que todos os que partilham a sessão alvo são reconfigurados;

- No segundo caso, pressupõe-se que o conjunto de canais definidos no âmbito é diferente do conjunto de canais que partilham a sessão. Neste cenário, os canais não afectados mantêm a sessão partilhada. Os restantes canais passam a ter um clone individual da sessão partilhada original. A acção de reconfiguração é aplicada a cada uma das sessões clone.

A Figura 4.5 ilustra esta situação. Na figura existe uma sessão d do tipo D, partilhada pelos canais x e y dos tipos X e Y, respectivamente. A acção pretende afinar o parâmetro p, atribuindo-lhe um novo valor cinco. No caso em que não se pretendem afectar todos os canais mas apenas o canal do tipo X, é declarado um âmbito de canal com o tipo em causa. Perante esta regra, a sessão partilhada é clonada. O canal não afectado mantém a sessão original e o canal alvo fica com a sessão clone. Após a aplicação da regra, apenas o canal do tipo X é afectado, com o parâmetro p afinado para cinco. No caso em que se pretende afectar todos os canais, não é definido âmbito de canal e após a aplicação da regra existe uma sessão partilha com o parâmetro afinado para o valor cinco.

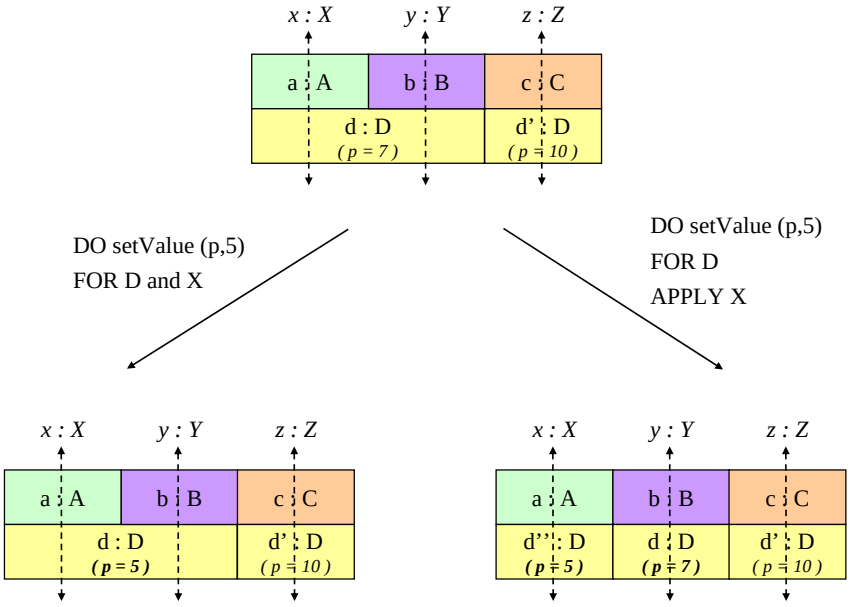

Figura 4.5: *setValue* em sessões partilhadas

O exemplo seguinte ilustra a declaração de uma regra cujas acções de reconfiguração são *setValue*:

```
WHEN BandwidthEvent: BandwidthEvent.value < MINBD
DO setValue(timeoutvalue,NEWVALUE) and
   setValue(congestionwindow,MINWINDOW)
   FOR TransportProtocol
```

O objectivo da regra apresentada é a alteração de dois parâmetros relevantes perante uma diminuição significativa da largura de banda. A regra é activada apenas pelo evento *BandwidthEvent* e a restrição à característica *value* inferior a MINBD. São declaradas duas acções, uma sem âmbito (de aplicação global) e outra com âmbito (de aplicação selectiva). A primeira acção altera o parâmetro *timeoutvalue* para *NEWVALUE*, afectando todos os protocolos que tenham o parâmetro, em todos os canais de todos os nós. A segunda acção afina o parâmetro *congestionwindow* para um valor *MINWINDOW*. Esta acção afecta todos os nós, dado que o âmbito topológico é omisso. No que toca aos protocolos, todos os que sejam tipos concretos do tipo *TransportProtocol* ou de subtipos, são afectados pela acção em causa. No caso de existirem sessões partilhadas por vários canais, todos os que têm a sessão na sua configuração são afectados.

4.3.4.2 *addProtocol*

A acção de reconfiguração *addProtocol* tem efeito a nível da configuração da pilha de protocolos. Nomeadamente, permite adicionar um protocolo numa posição determinada. O enriquecimento da qualidade de serviço através desta acção permite oferecer novas propriedades ou garantias. Por exemplo, adequando o serviço a novas condições de execução. A aplicação desta acção de reconfiguração pode ter diversas vantagens, de acordo com o contexto da aplicação. A acção aceita os três âmbitos possíveis e necessita de informação relativa a que protocolo adicionar e qual a sua posição. A sintaxe da acção é a seguinte:

$addProtocol(\text{newProtocol}, \text{position})$

O elemento *newProtocol* identifica qual o tipo concreto de protocolo a adicionar e *position* determina qual a sua posição relativa na configuração. Este último elemento aceita as partículas *above* ou *below*, que definem se a posição é acima ou abaixo do ponto de referência. O ponto de referência é definido pelo âmbito de protocolos, identificando quais as sessões que servem de referência para determinar a posição. Nas situações em que a ou as sessões de referência não se encontram na pilha, o protocolo não é adicionado. Nos casos em que existe mais do que uma sessão de referência na pilha, é adicionado um protocolo por cada

uma dessas sessões. Por fim, a declaração do âmbito topológico define quais os nós afectados e do âmbito de canais quais os canais alvo.

Quando as sessões de referência são partilhadas por vários canais, a aplicação da acção varia, consoante existe um âmbito de canais explícito ou não:

- No caso em que não existe um âmbito de canais explícito (ou se existe o conjunto de canais alvo é igual ao dos que partilham a sessão), a acção de reconfiguração afecta todos os canais que partilham a sessão de referência. É adicionada uma sessão a cada um desses canais;

- No segundo caso, com um âmbito explícito e um conjunto de canais alvo distinto do que partilha a sessão, são adicionadas instâncias individuais (não partilhadas) do protocolo apenas aos canais definidos no âmbito.

A Figura 4.6 ilustra os efeitos descritos. No primeiro caso, sem âmbito de canais, o protocolo é adicionado aos canais que partilham a sessão, x e y. O primeiro canal tem uma sessão e do tipo de protocolo E, ao passo que o segundo canal tem uma sessão distinta e'. As instâncias do protocolo E são iguais. No segundo caso, é adicionada apenas uma instância e do protocolo E no canal x, acima da sessão partilhada do tipo D.

O exemplo seguinte ilustra a declaração de uma acção de reconfiguração *addProtocol*:

```
WHEN ErrorEvent
DO addProtocol(DebugProtocol,above)
   WHERE ErrorEvent.NodeId
   FOR TransportProtocol
   APPLY DataChannel
```

A regra apresentada tem como objectivo adicionar um protocolo de *debug* à pilha de protocolos. A regra é despoletada e activada, dado que a primitiva *With* não é declarada, pelo evento *ErrorEvent*. A declaração da primitiva *Do* consiste numa acção de reconfiguração *addProtocol* e nos três âmbitos. A acção tem como parâmetros a identificação do protocolo a adicionar, *DebugProtocol*, e a posição relvativa, *above*. O âmbito de protocolos declarado na primitiva *For* determina que as sessões do tipo *TransportProtocol* são sessões referência.

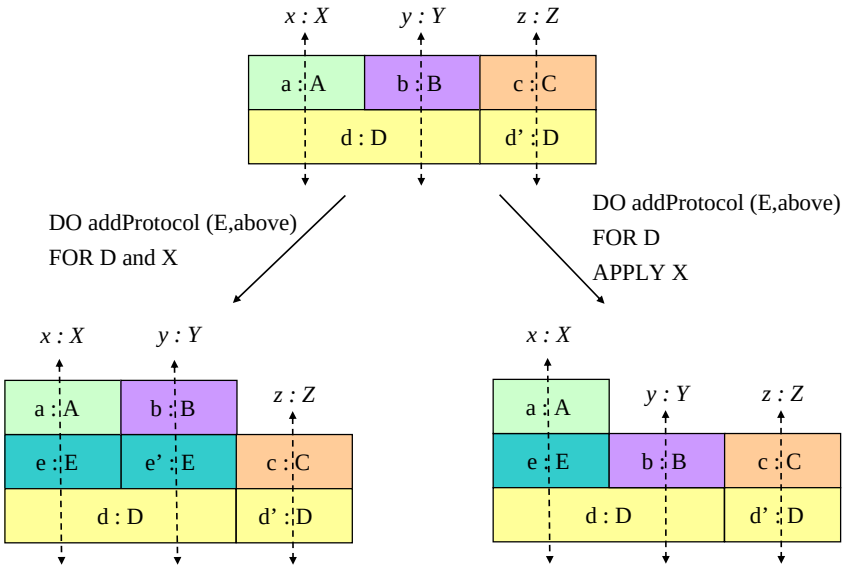

Figura 4.6: *addProtocol* em sessões partilhadas

Logo, as instâncias do protocolo serão adicionadas acima das sessões identificadas. O âmbito topológico determina que apenas o nó onde a alteração teve lugar é afectado pela acção. Por fim, o âmbito de canais define que, no caso das sessões de referência serem partilhadas, apenas os canais do tipo *DataChannel* sofrem a acção.

4.3.4.3 *removeProtocol*

A acção de reconfiguração *removeProtocol* tem efeito a nível da configuração da pilha de protocolos, tal como a acção *addProtocol*, permitindo remover um protocolo específico da pilha. Esta acção é útil para retirar protocolos que já não se adequam às condições de execução ou que prejudiquem o desempenho, dado que já não são necessários e continuam a consumir recursos. A necessidade e vantagens da aplicação desta regra dependem do contexto da aplicação. A acção não tem parâmetros e aceita os três âmbitos possíveis, contudo é o âmbito de protocolos que define qual o tipo de protocolos, cujas sessões devem ser retiradas. A remoção pode ser global ou selectiva. No primeiro caso todas as sessões são retiradas. No segundo caso, a utilização das partículas *above* ou *below* permite

seleccionar qual ou quais as instâncias a remover. Se não estiverem presentes instâncias do tipo de protocolo a retirar, a configuração da pilha não é afectada. No caso de não A sintaxe desta acção é a seguinte:

removeProtocol()

Tal como nas acções descritas anteriormente, a partilha de sessões afecta a aplicação da acção de reconfiguração, existindo dois efeitos possíveis:

- Se o âmbito de canais for omitido (ou o conjunto de canais que partilham a sessão é igual ao definido no âmbito), a sessão partilhada é retirada de todos os canais;
- No caso de um âmbito de canais distinto do conjunto de canais que partilham a sessão, apenas nos canais alvo é retirada a sessão.

A Figura 4.7 ilustra a situação com sessões partilhadas. Na figura pretende-se remover a sessão *d*. No primeiro caso, sem âmbito de canais, a sessão é retirada de todos os canais. No segundo caso, com âmbito de canais, apenas se pretende afectar o canal de tipo *X*. Nesta situação a sessão partilhada é separada em duas, e apenas a sessão do canal *x* é retirada.

O exemplo seguinte ilustra a remoção do protocolo adicionado no exemplo da acção *addProtocol*, na Secção 4.3.4.2:

```
WHEN ErrorEndEvent
DO removeProtocol()
   WHERE {ErrorEndEvent.nodeId}
   FOR DebugProtocol above TransportProtocol
   APPLY DataChannel
```

A regra declarada é despoletada pelo evento *ErrorEndEvent*, que indica o fim da situação incorrecta detectada. A inexistência de declaração da primitiva *With* significa que quando a regra é despoletada, é também activada. A declaração da primitiva *Do* consiste na acção *removeProtocol* e nos três âmbitos. O âmbito topológico define que apenas o nó onde ocorre a situação incorrecta é afectado. O âmbito de protocolos declara que o protocolo a ser removido é do tipo *DebugProtocol* e a sua localização encontra-se acima (*above*) das sessões que sejam instâncias de protocolos do tipo *TransportProtocol*. Por fim, o âmbito de canais

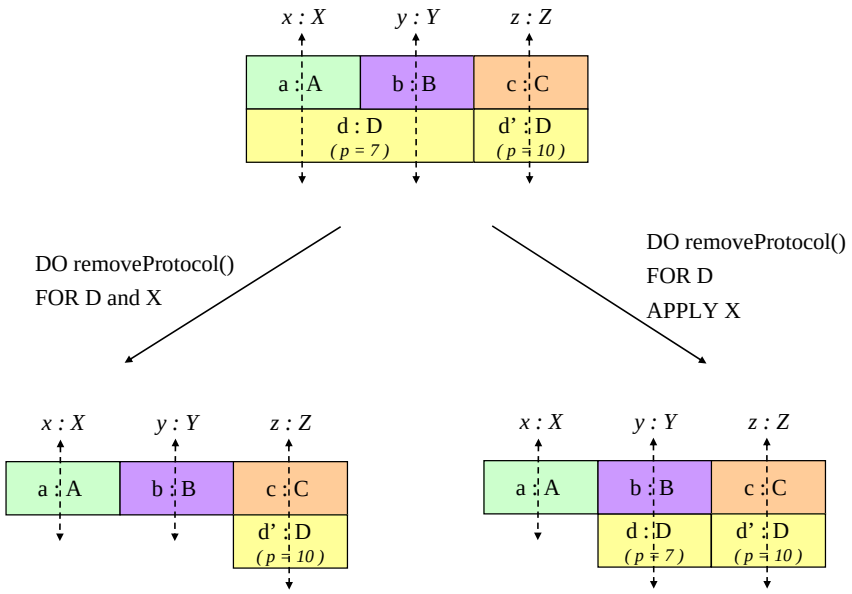

Figura 4.7: *removeProtocol* em sessões partilhadas

determina que apenas os canais do tipo *DataChannel* são afectados, se existirem sessões partilhadas.

4.3.4.4 *changeProtocol*

A acção de reconfiguração *changeProtocol* é a junção das acções *removeProtocol* e *addProtocol*. Esta acção altera a configuração da pilha de protocolos, sem alterar o número de protocolos na pilha. Nas situações em que existem várias concretizações distintas de um protocolo, esta acção é útil, pois permite uma permuta entre diferentes concretizações. A acção visada é vantajosa quando se pretende substituir uma concretização por outra mais adequada às condições de execução, optimizando o desempenho no contexto da aplicação. Em situações em que é necessário substituir uma garantia por outra, de forma a manter a correcção do serviço, esta acção também é adequada. A acção aceita um parâmetro que define qual o novo protocolo, bem como os três âmbitos que controlam o conjunto de alvos. A sintaxe é a que se segue:

$changeProtocol(\mathrm{newProtocol})$

O parâmetro aceite, *newProtocol*, é um tipo de protocolo concreto que identifica qual o novo protocolo a pertencer à configuração da pilha. O conjunto de alvos da substituição é definido no âmbito de protocolos, através da declaração do tipo de protocolos, abstracto ou concreto, e ainda, se necessário, a localização relativa na pilha de protocolos através das partículas *above* e *below*. Tal como nas acções *addProtocol* e *removeProtocol*, a inexistência de sessões dos protocolos alvo resulta na manutenção da pilha de protocolos e, no caso de várias instâncias, todas são substituídas.

Considerando a situação de sessões partilhadas, o efeito da acção de permuta tem dois efeitos possíveis

- Num deles, a substituição ocorre em todos os canais que partilham a sessão, dado que é omitido o âmbito de canais ou o âmbito é igual ao conjunto de canais que partilham a sessão. A substituição mantém a partilha;
- O outro efeito, implica a substituição apenas no conjunto definido pelo âmbito de canais (considerando que é distinto do conjunto de canais que partilham a sessão). A substituição mantém a partilha no conjunto de canais afectados.

Os efeitos descritos são ilustrados pela Figura 4.8. Nesta figura, o protocolo do tipo D vai ser substituído pelo protocolo de tipo concreto E. A sessão d é partilhada pelos canais de tipo X e Y. Na primeira situação, omissão de âmbito de canais, a sessão do tipo de protocolo D é substituída nos canais que a partilham. A nova sessão e é partilhada por ambos os canais x e y. Na segunda situação, com declaração do âmbito de canais, apenas o canal x é afectado pela acção, logo o canal y mantém a sessão original e em x esta é substituída pela nova sessão e do tipo de protocolo E.

O exemplo seguinte ilustra a declaração de uma regra cuja acção de reconfiguração é *changeProtocol*:

```
WHEN AllWiredEvent
WITH Network.numberOfNodes > 3
DO changeProtocol(TotalOrderProtocol)
   FOR OrderProtocol
   APPLY DataChannel
```

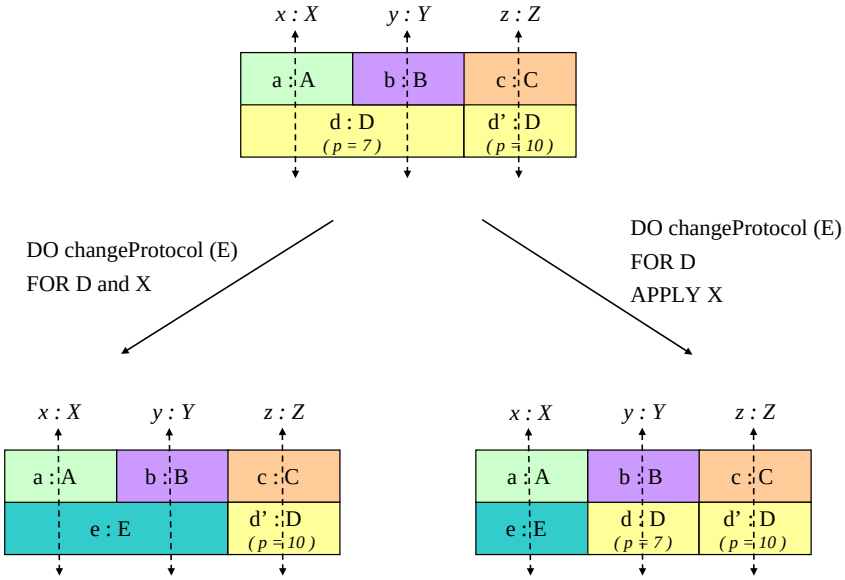

Figura 4.8: *changeProtocol* em sessões partilhadas

Neste exemplo, a regra declarada é despoletada pelo evento *AllWiredEvent*. A regra apenas é activada se a declaração *With* for cumprida, ou seja, o número de nós da rede for superior a três. A declaração da primitva *Do* define que o tipo concreto de protocolo *TotalOrderProtocol* deve substituir todos os protocolos do tipo *OrderProtocol*, sem restrições à localização destes protocolos na pilha. A omissão de âmbito topológico indica que esta acção afecta todos os nós da rede. No caso de sessões partilhadas, apenas as dos canais do tipo *DataChannel* são substituídas.

4.3.4.5 *changeQoS*

A última acção de reconfiguração, *changeQoS*, tem efeitos na configuração da pilha de protocolos, substituindo-a por uma nova configuração. Esta substituição não exige que a nova configuração tenha igual número de protocolos ou aplica restrições à composição dos protocolos. A utilização desta acção de reconfiguração normalmente implica que a adaptação a realizar é mais complexa do que com as acções descritas anteriormente. Portanto, a acção é útil em situações em que seja necessário mudar por completo a configuração, devido a uma mudança radical

nas condições de execução. Relativamente à sintaxe, esta acção apenas aceita um parâmetro, a nova configuração da pilha. A nova configuração é uma sequência de tipos concretos de protocolos, ordenados desde o protocolo base da pilha até ao de topo. A sintaxe da acção de reconfiguração é a seguinte:

changeQoS(newQoS)

O parâmetro *newQoS* é a sequência de protocolos que constitui a nova configuração. A acção apenas aceita os âmbitos topológicos e de canais. O âmbito de protocolos não é aceite. Ao contrário das acções de reconfiguração descritas nas secções anteriores, a granularidade da acção de reconfiguração visada é menor. Assim, o elemento com maior granularidade com o qual esta acção lida são os canais, ao invés dos protocolos.

Dado o efeito da acção de reconfiguração visada, não existem diferentes resultados possíveis perante sessões partilhadas. A substituição de uma pilha de protocolos com uma sessão partilhada, implica que a sessão se mantém inalterada nos canais não afectados e desaparece na nova configuração.

Esta acção é ilustrada pelo exemplo que se segue:

```
WHEN NewConnectivityEvent: NewConnectivityEvent.wired
WITH Network.isWired(NewConnectivityEvent.NodeId)
DO changeQoS(WiredQoS)
   WHERE {NewConnectivityEvent.NodeId}
   APPLY DataChannel

WHEN NewConnectivityEvent:!NewConnectivityEvent.wired
WITH !Network.isWired(NewConnectivityEvent.NodeId)
DO changeQoS(WirelessQoS)
   WHERE {NewConnectivityEvent.NodeId}
   APPLY DataChannel
```

O exemplo tem por base um cenário em que os dispositivos podem ter ligação com fios ou sem fios, existindo uma configuração específica da pilha de protocolos para cada uma das situações. As diferentes configurações são as seguintes:

```
WiredQoS = [TCPTranspProtocol, WiredProtocol, ...]
WirelessQoS = [UDPTranspProtocol, WirelessProtocol, ...]
```

A primeira configuração, denominada *WiredQoS* é a utilizada na ligação com fios, ao passo que a sequência declarada em *WirelessQoS* é a utilizada na li-

gação sem fios. As sequências não estão completas e são apresentadas a título meramente ilustrativo. No exemplo alterna-se entre estas duas configurações.

Ambas as regras são despoletadas por um evento *NewConnectivityEvent* com restrições. A primeira regra diz respeito à ligação com fios, dado que a restrição à informação contida no evento obriga a que a nova ligação seja com fios, *wired*. A primitiva *With* declara a condição imposta para que a regra seja activada. A condição é que a actual configuração da pilha seja a *WirelessQoS* no nó que originou o evento. Esta condição é declarada através da verificação da característica observável *isWired* no *contextType Network*. Após a activação da regra, a acção declarada *changeQoS* obriga à substituição da configuração da pilha pela *WiredQoS*. O âmbito topológico da acção é apenas um nó, identificado como o nó que originou o evento. Por fim, o âmbito de canais indica que apenas os canais do tipo *DataChannel* são afectados. A segunda regra indica precisamente a situação inversa. Perante uma nova ligação, sem fios, em que a configuração da pilha no nó visado é a *WiredQoS* realiza-se a acção de substituir a configuração pela *WirelessQoS*. Os nós e canais afectados são os mesmos da primeira regra.

4.4 Estratégias para Aplicar a Reconfiguração

As políticas definem a lógica da adaptação, indicando quais as acções de reconfiguração a efectuar em cada situação mas não a forma como essas acções são aplicadas em cada sistema em concreto. Assim, a forma como as acções são aplicadas é definida exteriormente à política.

As estratégias para aplicar a reconfiguração definem como são aplicadas as acções de reconfiguração. Uma estratégia consiste no conjunto de passos necessários para aplicar uma acção. As preferências do programador, as propriedades do sistema, as características da rede e a natureza da acção de reconfiguração determinam qual a estratégia mais adequada a cada caso.

Tipicamente, uma estratégia é constituída por várias fases. Essas fases correspondem às necessidades das diferentes acções de reconfiguração. Estas necessidades podem ser: a paragem do envio de mensagens, a coordenação entre nós, a recolha e recuperação de estado, entre outras. Destas necessidades, o grau de coordenação entre os nós é aquela que diferencia profundamente as estratégias.

Assim, é possível distinguir as estratégias em globais e locais (Grace *et al.*, 2006). As estratégias locais não implicam coordenação entre os nós alvo da acção de reconfiguração. Por outro lado, as estratégias globais, exigem coordenação entre todos os nós. As estratégias locais e globais correspondem a extremos de coordenação entre os nós. Entre estes extremos é possível definir uma miríade de estratégias com necessidades de coordenação intermédias.

De seguida são abordadas as estratégias locais e globais em pormenor, descrevendo-se qual a estratégia utilizada por cada uma das acções de reconfiguração descritas na Secção 4.3.4.

4.4.1 Estratégias Locais

As estratégias locais não exigem coordenação entre os nós para aplicar a acção de reconfiguração, mesmo quando esta tem como alvo diversos nós. A especificação de uma estratégia local exclui quaisquer instruções relativas à coordenação. Os passos definidos dependem de estratégia para estratégia, contudo a ausência de coordenação pressupõe três fases comuns. A primeira fase é informar os nós alvo da acção ou acções de reconfiguração que têm de aplicar. A segunda fase será a aplicação da acção nos nós alvo. Finalmente, a terceira fase é a confirmação dos nós em como concluíram com sucesso a reconfiguração. Cada fase pode ser constituída por um ou mais passos.

As vantagens das estratégias locais prendem-se com a ausência de coordenação e, consequente, com a sua simplicidade, relativamente às globais. A ausência de coordenação permite um desempenho superior, reflexo da independência dos outros nós alvo. Esta independência significa que os nós alvo suspendem a sua actividade apenas durante o intervalo de tempo estritamente necessário para aplicar a reconfiguração.

Qualquer acção pode ser aplicada utilizando uma estratégia local. No entanto, uma estratégia local pode não oferecer as garantias de coerência necessárias. As estratégias locais são adequadas para situações sem necessidade de coordenação entre nós, ou por ser apenas um nó, ou porque a simultaneidade de aplicação é irrelevante. Noutras situações, optar por uma estratégia local depende simplesmente do desempenho.

Um exemplo concreto de uma estratégia local consiste nos seguintes passos. Um gestor da adaptação envia a acção de reconfiguração para os nós alvo. Os nós recebem a acção e aplicam-na. Após a conclusão da reconfiguração, os nós confirmam ao gestor que terminaram a acção com sucesso.

A acção de reconfiguração *setValue*, do trabalho desenvolvido, permite alterar um parâmetro de um protocolo ou protocolos. A acção é aplicada pela estratégia local descrita anteriormente. A ausência de coordenação permite aplicar a acção com um desempenho óptimo, sem existirem esperas por outros nós alvo. A estratégia não oferece a garantia de que todos os nós alvo concretizem a reconfiguração ao mesmo tempo. A única garantia oferecida é a da reconfiguração ter sido concretizada.

4.4.2 Estratégias Globais

As estratégias globais, ao contrário das locais, utilizam a coordenação entre os nós para aplicar a acção de reconfiguração. A especificação de uma estratégia global inclui os passos que asseguram a coordenação entre os nós. As estratégias globais pressupõem as mesmas três fases das estratégias locais: preparação, reconfiguração e confirmação. No entanto, as fases de preparação e confirmação têm passos adicionais, relacionados com a coordenação.

As vantagens das estratégias globais prendem-se com a coordenação e as garantias que estas oferecem. Os requisitos de coordenação, apesar de introduzirem atrasos e prejudicarem o desempenho, oferecem garantias de simultaneidade.

Tal como as estratégias locais, as estratégias globais podem aplicar qualquer acção de reconfiguração, mesmo que seja em apenas um nó. No entanto, estas estratégias são orientadas para acções cujo âmbito topológico é um conjunto de nós. A coordenação é vantajosa nas situações em que todos os nós devem atingir um estado quiescente antes da aplicação da reconfiguração. A reconfiguração é atrasada até que todos os nós alvo estejam preparados. A mesma necessidade de coordenação também se verifica quando é necessário que todos os nós recomecem a sua actividade ao mesmo tempo.

Um exemplo concreto de uma estratégia global, com manutenção de estado, consiste nos passos seguintes. A aplicação da estratégia é coordenada por uma

entidade centralizada, o gestor de adaptação. Em primeiro lugar, o gestor da adaptação convoca os nós alvo de reconfiguração para atingirem um estado quiescente. Cada nó prepara-se, recolhe o estado e confirma estar preparado. O gestor, após a recepção de todas as confirmações, envia a acção de reconfiguração. Cada nó efectua a acção de reconfiguração recebida. Após a sua conclusão, recupera o estado anterior à reconfiguração e confirma ao gestor. Após todos os nós alvo estarem prontos, o gestor dá permissão para continuar a actividade.

As acções de reconfiguração *addProtocol*, *removeProtocol*, *changeProtocol* e *changeQoS* utilizam estratégias globais, em concreto a descrita acima. Estas acções afectam a qualidade de serviço e a configuração das pilhas de protocolos. Se as alterações não forem realizadas simultaneamente, não é garantido o correcto funcionamento dos nós do sistema. Por esta razão, as estratégias globais são as mais adequadas, oferecendo as garantias de coerência necessárias.

Sumário

Neste capítulo foram apresentados os modelos de contexto, os modelos de alvos da adaptação e a linguagem de especificação de políticas de adaptação. A linguagem prevê diferentes acções de reconfiguração, concretizadas por estratégias para aplicar a reconfiguração, as quais também foram descritas.

No capítulo seguinte é apresentado o *proof of concept* da arquitectura descrita no Capítulo 3, que ilustra a aplicação dos elementos apresentados neste capítulo.

Capítulo 5

Concretização

Neste capítulo, é descrita a concretização de um protótipo que segue as considerações da arquitectura apresentada no Capítulo 3. O protótipo foi desenvolvido numa plataforma *Appia* modificada, cujo enriquecimento é descrito de seguida.

5.1 Enriquecimento da Plataforma *Appia*

A plataforma *Appia*, introduzida na Secção 2.3.2.2, permite compor serviços de comunicação com base numa biblioteca de protocolos diversificada que pode ser expandida. Os protocolos distinguem-se pelo seu nome, seguindo interfaces bem definidas. A interface do protocolo especifica quais os eventos que este aceita, necessita ou fornece. Uma composição de instâncias protocolos, um canal, oferece um serviço com propriedades que advêm dos protocolos que a constituem e pela sua ordem na pilha protocolar. Os canais são referidos pelo seu nome e a sua composição é estática a partir do momento da sua criação.

Esta abordagem estática relativa à composição dos canais, juntamente com a inexistência de suporte a tipos e hierarquias a nível dos protocolos e canais, levanta vários obstáculos ao suporte à adaptação, tornando a plataforma inadequada para a reconfiguração dinâmica. Para contornar estas limitações a plataforma foi munida de suporte a tipos e hierarquias de canais e protocolos. Foi igualmente proposta uma técnica para alterar dinamicamente a qualidade de serviço dos canais, tendo sido desenvolvido um conjunto de agentes que a executam.

Esta técnica permite superar as dificuldades resultantes da composição estática dos canais.

Neste capítulo, as referências à plataforma *Appia* consideram a versão 2.1, a mais recente distribuição disponível aquando do início do desenvolvimento do protótipo. As extensões propostas à plataforma são designadas por *Reconfigurable Appia* (*RAppia*).

5.1.1 Hierarquia de Tipos de Protocolos

Os protocolos distinguem-se pelas propriedades que imprimem ao fluxo de mensagens. A biblioteca de protocolos da plataforma *Appia* engloba protocolos de ordenação, difusão, transporte, entre outros. A tipificação dos canais passa por atribuir tipos consoante as propriedades de cada protocolo.

A atribuição de tipos baseia-se na utilização de interfaces da linguagem *Java*. Foram definidas várias interfaces de acordo com a biblioteca de protocolos. Cada interface representa um tipo abstracto ou um tipo concreto. A organização dos tipos de protocolos numa hierarquia passa pela utilização de outro mecanismo da linguagem *Java*, a herança. Todas as interfaces herdam de uma interface mãe *Protocol*, que corresponde ao tipo abstracto raíz. São definidas as interfaces dos restantes tipos de protocolos abstractos e concretos e é construída a herança entre eles. Desta forma, existe sempre uma interface que representa um tipo concreto de protocolo. Cada protocolo apenas implementa a interface correspondente ao seu tipo concreto.

A definição de um protocolo na plataforma *Appia* passa por especificar duas classes. Uma delas estende a classe original *Layer* (camada) e a outra estende a classe *Session* (sessão). A primeira define quais os eventos necessários, aceites e fornecidos pelo protocolo. A segunda define como são tratados esses eventos, definindo o comportamento do protocolo. Na plataforma *Appia*, a definição de um protocolo de difusão fiável é a seguinte:

```
public class ReliableMulticastLayer extends Layer{
...
}
public class ReliableMulticastSession extends Session{
...
```

```
}
```

A tipificação deste protocolo passa por implementar a interface correspondente ao seu tipo concreto. Dado que a herança da linguagem *Java* já define a hierarquia entre interfaces, basta apenas implementar a interface do tipo concreto, para que, automaticamente, sejam implementadas as interfaces de todos os tipos abstractos de que deriva. A definição da interface do tipo concreto *ReliableMulticastProtocol* resultou do seguinte:

```
public interface Protocol{}

public interface MulticastProtocol extends Protocol{}

public interface ReliableMulticastProtocol extends MulticastProtocol
    {}
```

Foi criada a interface *Protocol*, estendida pela interface *MulticastProtocol* e, por fim, estendida por *ReliableMulticastProtocol*. Com base na descrição anterior da interface, a nova definição do protocolo em *RAppia* é:

```
public class ReliableMulticastLayer extends Layer implements
    ReliableMulticastProtocol{
...
}
public class ReliableMulticastSession extends Session{
...
}
```

Note-se que apenas a extensão à classe *Layer* é contemplada para implementar a interface. A sessão mantém-se inalterada. Assim, este protocolo passa a pertencer a um determinado patamar da hierarquia de tipos de protocolos da plataforma *RAppia*. A Figura 5.1 ilustra parcialmente a hierarquia construída para a plataforma. Os protocolos abstractos estão representados a itálico, ao passo que os protocolos concretos estão em texto normal.

A hierarquia apresentada na figura tem como raíz o tipo abstracto *Protocol*. Deste tipo são subtipos abstractos: *MulticastProtocol*, *OrderProtocol* e *UtilProtocol*. O primeiro refere-se a protocolos que ofereçam difusão de mensagens, o segundo trata protocolos de ordenação de mensagens e o terceiro engloba protocolos que realizam outras funções como *logging* ou contagem de mensagens, por

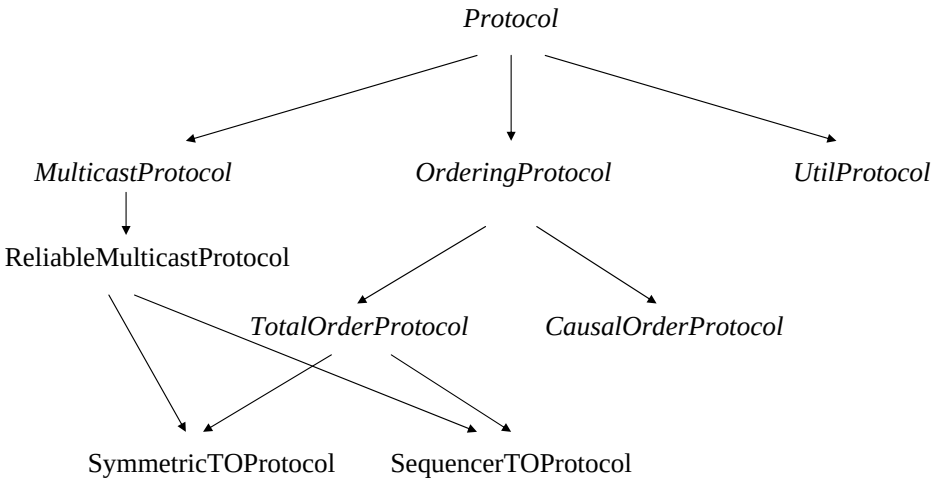

Figura 5.1: Hierarquia de Tipos de Protocolos

exemplo. O tipo *MulticastProtocol* tem um subtipo concreto denominado *ReliableMulticastProtocol*, que oferece fiabilidade. O tipo *OrderingProtocol* tem dois subtipos abstractos denominados *TotalOrderProtocol* e *CausalOrderProtocol*. O primeiro refere-se a ordenação total de mensagens, ao passo que o segundo a ordenação causal. Na hierarquia apresentada na figura, o tipo abstracto *TotalOrderProtocol* apresenta um subtipo concreto *SymmetricTOProtocol*, que é igualmente subtipo concreto de *ReliableMulticastProtocol*.

5.1.2 Hierarquia de Tipos de Canais

A construção de tipos e de uma hierarquia de canais não segue, exactamente, a aproximação baseada nas propriedades, utilizada para os protocolos. Tal deve-se ao facto dos canais serem definidos por uma composição de protocolos, o que resulta em dois aspectos. O primeiro aspecto prende-se com o facto da composição dos canais evoluir ao longo do tempo, por meio de reconfigurações. O segundo aspecto deve-se ao facto de composições iguais poderem resultar em canais com funções distintas. Assim, um canal não pode ser tipificado em função das suas propriedades, ou seja, da sua composição de protocolos.

Na *RAppia*, a aproximação utilizada para tipificar os canais baseou-se na função que estes desempenham a nível da aplicação. A função desempenhada

por um canal não varia com tempo. Em contrapartida, esta função determina a sua reconfiguração. Assim, os tipos de canais são atribuídos pela aplicação.

Um tipo de canal, abstracto ou concreto, é representado por uma interface da linguagem *Java* e a hierarquia de tipos é baseada na herança. Todas as interfaces herdam de uma interface mãe *Channel*, que corresponde ao tipo abstracto raíz. A definição das restantes interfaces baseia-se na extensão da interface mãe.

Na plataforma *Appia* uma aplicação define um canal com base numa qualidade de serviço (composição de protocolos) e atribuindo-lhe um nome. A definição de um canal é ilustrada pelo excerto seguinte de um exemplo:

```
public class Application{
 private static Layer[] qos={
    new appia.protocols.udpsimple.UdpSimpleLayer(),
    new ApplicationLayer()
  };
 QoS myQoS = new QoS("AudioDataQoS",qos);
 Channel myChannel = myQoS.createUnboundChannel("AudioDataChannel");
...
}
```

O exemplo descrito passa por criar uma composição de protocolos, constituída por dois protocolos *UdpSimple* e *Application*. A partir desta composição é criada a qualidade de serviço correspondente e, por fim, é criado o canal de nome *AudioDataChannel*.

Note-se que, na plataforma *Appia* todos os canais são instâncias criadas a partir da mesma classe *Channel* e, desta forma, todos os canais possuem exactamente o mesmo tipo. Para contornar esta limitação, na plataforma *RAppia*, os canais especificam o seu tipo através de um atributo da classe que define o nome da interface. O excerto de código seguinte descreve o enriquecimento da classe para suportar tipos:

```
public class Channel{

private Class myChannelType = null;

public void setType(Class newchanneltype){
        myChannelType = mynewtype;
}
```

```
public Class getType(){
        return myChannelType;
}
...
}
```

Retomando o exemplo da definição de um canal, pela aplicação, a nova definição obriga, primeiro, a definir o conjunto de interfaces necessárias e só depois adicionar o tipo de canal. A definição das interfaces para o tipo concreto *AudioMediaChannel* seria a seguinte:

```
public interface Channel{}

public interface MediaChannel extends Channel{}

public interface AudioMediaChannel extends MediaChannel{}
```

Assim, já é possível definir o novo canal. A atribuição do tipo passa por invocar o método *setType* da classe *Channel*, com o nome da interface que representa o tipo concreto, como é ilustrado de seguida:

```
public class Application{
...
 Channel myChannel = myQoS.createUnboundChannel("AudioDataChannel");
 myChannel.setType(appia.types.channels.AudioMediaChannel.class);
...
}
```

Uma hierarquia parcial de tipos de canais é ilustrada pela Figura 5.2. Os tipos abstractos são representados a itálico e os concretos a texto normal.

O tipo abstracto *Channel* tem três subtipos abstractos: *ControlChannel*, *DataChannel* e *UpdateChannel*. O tipo abstracto *DataChannel*, referente a transmissão de dados, tem um subtipo abstracto *MediaChannel* com três tipos concretos: TextMediaChannel, VideoMediaChannel e AudioMediaChannel. O primeiro refere-se a conteúdo texto, o segundo a vídeo e o último a áudio.

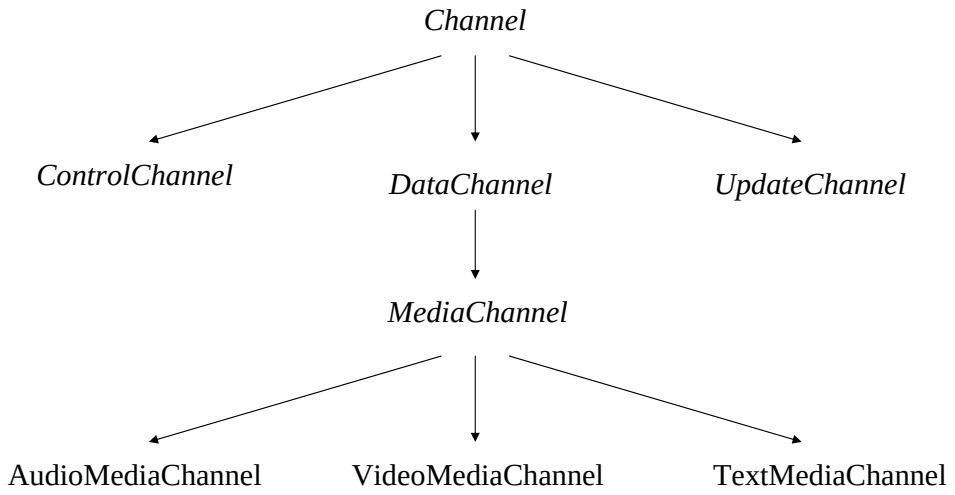

Figura 5.2: Hierarquia de Tipos de Canais

5.1.3 Técnica para Alteração Dinâmica da Composição

A técnica para alteração dinâmica da composição de protocolos de um canal permite ultrapassar os obstáculos colocados pela plataforma *Appia*, a nível da ausência de suporte para a alteração da pilha de protocolos após a criação do canal.

A inexistência deste suporte prende-se com a associação estática entre canal e qualidade de serviço utilizada pela plataforma. Esta associação é estática porque a plataforma gera os trajectos a percorrer por cada evento no início da execução. Alterações à composição dos canais, implicariam recalcular os trajectos.

A solução óbvia para este problema passaria por munir a plataforma desse suporte e alterando-a de forma a recalcular o trajecto dos eventos sempre que são efectuadas alterações à composição. No entanto, para tal, seriam necessárias alterações ao núcleo da plataforma. A realização dessas alterações ultrapassa o âmbito deste trabalho e exigiria conhecimento profundo da construção interna da plataforma. Assim, foi utilizada outra solução.

Esta solução passa pela criação de um novo canal, cuja composição reflecte a nova reconfiguração desejada e a transferência do estado do canal anterior para o novo, substituindo o canal de forma quase transparente para a aplicação. No

entanto, esta solução apresenta várias vantagens e desvantagens. A principal vantagem é, sem dúvida, evitar a remodelação da plataforma, poupando tempo e trabalho, bem como é dada a garantia de que a extensão *RAppia* continua compatível com aplicações já desenvolvidas. Contudo, a criação de um novo canal, substituto, implica vários cuidados. Estes cuidados prendem-se com o estado de cada um dos protocolos, com o processamento em curso de eventos e com possíveis eventos armazenados pelos protocolos. Esta solução implica ainda que o fluxo de mensagens seja interrompido temporariamente, o que pode prejudicar o desempenho.

As várias dificuldades são tidas em consideração pela solução proposta. O estado de uma instância de um protocolo, uma sessão, está relacionado com a informação vital que resulta da execução do protocolo. O estado pode referir-se, por exemplo, aos valores que atributos chave tomam, como os números de sequência atribuídos às mensagens. Se for criada uma nova instância do protocolo, o número de sequência inicial é diferente do dado pela execução da sessão anterior. Assim, é necessário transferir o estado do canal anterior para o novo canal. Para isso, a *RAppia* obriga a que os protocolos identifiquem a informação relativa ao estado (que é necessário transferir) e que sejam capazes de recuperar essa informação quando fornecida externamente (nomeadamente pelo agente de reconfiguração).

Relativamente ao processamento em curso de eventos, quando é recebida uma ordem de reconfiguração, é necessário que todas as sessões do canal atinjam um estado quiescente. Esse estado implica que todos os eventos sejam processados, incluindo os eventos armazenados nas sessões. O estado quiescente implica também que as sessões não gerem mais eventos.

Esta técnica, juntamente com o suporte para o tratamento dos problemas acima descritos, foi concretizada no âmbito do trabalho apresentado nesta dissertação. A técnica é utilizada nas acções de reconfiguração que afectam a composição dos canais e a sua realização passou pela construção de um agente de reconfiguração que executa diversas directivas e utiliza a técnica para reconfigurar os canais. Desta forma foi igualmente desenvolvido um conjunto de instruções orientadas às estratégias desenvolvidas, mas suficientemente genéricas para serem

utilizadas noutras estratégias. A descrição destes aspectos é relatada em pormenor na Secção 5.2.2.2, relativa aos agentes de reconfiguração.

5.2 Concretização do Protótipo

Com as extensões à plataforma *Appia*, descritas anteriormente, e que se designa por *RAppia* é possível concretizar os restantes componentes da arquitectura. Todos os componentes são materializados sob a forma de protocolos únicos. Os componentes locais aos nós não afectam a qualidade de serviço e reúnem-se num canal específico para o domínio da adaptação. De seguida é apresentada a concretização do protótipo, de acordo com a separação definida na descrição da arquitectura. Em primeiro lugar serão referidos os componentes de contexto e posteriormente os componentes de adaptação.

5.2.1 Componentes de Contexto

Os componentes de contexto englobam os sensores e o monitor de contexto e são responsáveis por capturar, monitorizar, armazenar e assinalar informação de contexto relevante. Para isso, os sensores de contexto são responsáveis por capturar a informação e publicá-la para o monitor. O monitor interpreta, armazena e assinala as alterações de contexto relevantes. A Figura 5.3 ilustra os componentes de contexto, descritos de seguida.

5.2.1.1 Sensores de Contexto

Os sensores capturam informação de contexto, nomeadamente a informação observável (*sensed observable*), que pode ser recolhida directamente. Os sensores não interpretam informação, função da responsabilidade do monitor de contexto. Na plataforma *RAppia*, a interface disponibilizada é uma de composição de protocolos, implicando que todas as aplicações sejam concretizadas sob a forma de um protocolo. Assim os sensores de contexto desenvolvidos foram concretizados como protocolos, integrados nos canais que executam no nó, e não afectam as

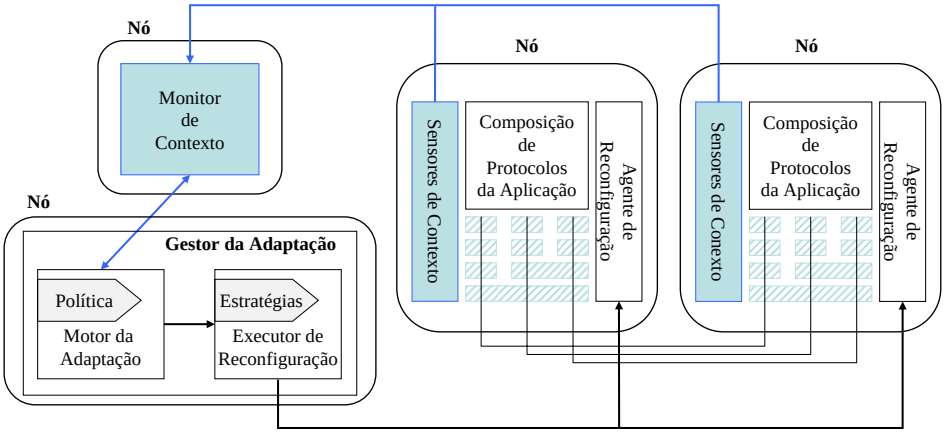

Figura 5.3: componentes de Contexto

propriedades do serviço. Os protocolos que concretizam sensores estão presentes em todos os canais monitorizados. No caso de um sensor monitorizar vários canais, a sessão do sensor é partilhada pelos canais.

O sensores podem ser especializados ou genéricos. Os primeiros recolhem apenas um tipo de informação específica, ao passo que os segundos, podem recolher diversos tipos de informação de contexto. Nesta perspectiva, em cada nó, existe um ou mais sensores de contexto locais.

Os sensores apenas capturam informação observada no nó onde estão instalados. A captura de informação pode ser efectuada periodicamente, de acordo com um valor estipulado, ou em resposta a estímulos do ambiente, como na presença de um determinado evento. No primeiro caso, a informação é capturada numa abordagem pergunta-resposta. Um sensor especializado envia um pedido de informação e a sessão responde com a informação. Esta informação é encaminhada pelo sensor para o monitor de contexto. Se for um sensor genérico, este pode efectuar vários pedidos distintos, esperar as várias respostas e construir informação concreta, em vez de a encaminhar em bruto para o monitor de contexto. Os pedidos, respostas e encaminhamento para o monitor são realizados através de eventos. A Figura 5.4 descreve os eventos trocados pelos sensores no caso de pedido-resposta.

A indicação de início da captura é marcada pelo símbolo T, que indica o

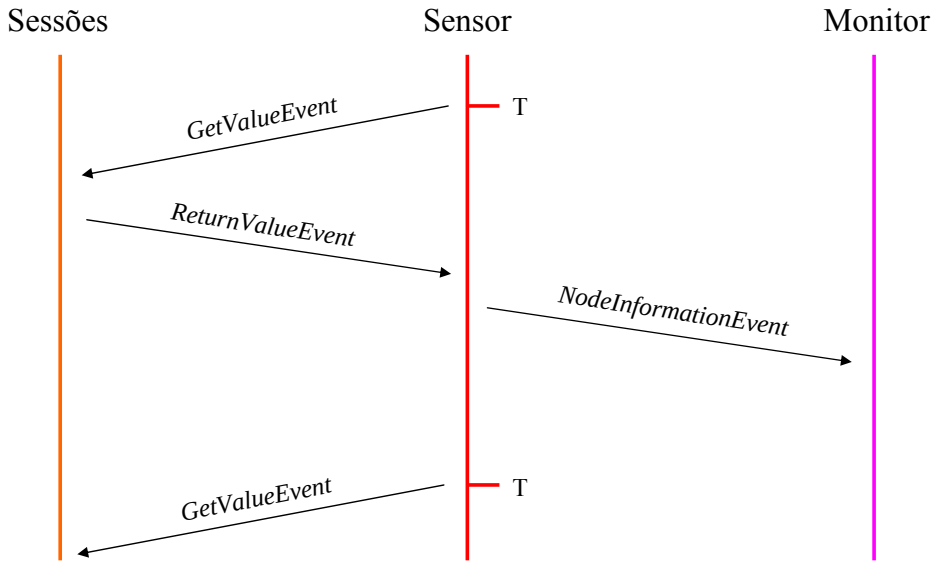

Figura 5.4: Dinâmica de eventos entre os sensores, sessões e monitor de contexto

período. A cada período é enviado um evento do sensor para os canais do nó, a fim de recolher informação. O evento denomina-se *GetValueEvent*. As sessões que aceitam o evento respondem com os valores pedidos, através de um evento *ReturnValueEvent*. O sensor, após recepção dos valores pedidos, envia um evento *NodeInformationEvent* para o monitor de contexto, actualizando a informação do nó onde se encontra. Passado o período de tempo estipulado, da primeira captura, o processo repete-se.

Para os sensores que respondem a estímulos do ambiente, o excerto de código seguinte exemplifica a definição de um sensor sensível à alteração da taxa de erros no envio de mensagens, notificada pelo evento *ErrorEvent*.

```
public class NodesSensorLayer extends Layer{
 public SensorLayer() {
  super();
  evProvide = new Class[]{
   NodeInformationEvent.class,
  };
  evAccept = new Class[]{
   ErrorEvent.class
  };
```

```
  evRequire = new Class[]{};
        }
}
public class SensorSession extends Session{
 double errorRate;

 public void handle(Event e){
  if(e instanceof ErrorEvent)
   handleError((ErrorEvent)e);
          ...
 }

 private void handleError(ErrorEvent event) {
        double newErrorRate = event.errorrate;
        errorRate = newErrorRate;
        NodeInformationEvent nievent = new NodeInformationEvent(
            errorRate);
        ...
        event.go();
        }
 ...
}
```

Neste exemplo, o sensor envia um evento ao monitor de contexto, sempre que a taxa de erros no nó se altera, num evento *NodeInformationEvent*, que transporta o valor da taxa de erros. Este sensor tem a capacidade de receber periodicamente eventos *ErrorRate*, bem como fazer um pedido de informação aos protocolos implicados, que respondem com os eventos em questão.

5.2.1.2 Monitor de Contexto

O monitor de contexto é responsável pela concretização do modelo de contexto, assegurando que a informação de contexto especificada está disponível e que os eventos de contexto são lançados. Tal como os sensores, o monitor de contexto é concretizado sob a forma de um protocolo. Uma instância do protocolo corre num nó que não oferece serviços, sendo apenas para a função de suporte à adaptação.

O monitor depende dos sensores de contexto para obter informação actualizada que pode ser disponibilizada sem processamento adicional ou, então, inter-

pretada. Quer a informação recolhida directamente dos sensores, quer a informação obtida através da interpretação destes dados, é armazenada pelo monitor de contexto para posterior consulta. O acesso à informação armazenada é conseguido através de interrogações realizadas pelo gestor da adaptação ao monitor.

O monitor é também responsável por gerar os eventos definidos no modelo de contexto. Estes eventos assinalam alterações de contexto relevantes. Na recepção de informação de contexto nova, o monitor analisa-a para determinar se existem alterações relevantes. Se existirem, notifica o gestor da adaptação gerando os eventos definidos no modelo de contexto.

O monitor de contexto troca eventos com os sensores e com o gestor da adaptação. O fluxo de ambos os tipos de eventos é descrito na Figura 5.5.

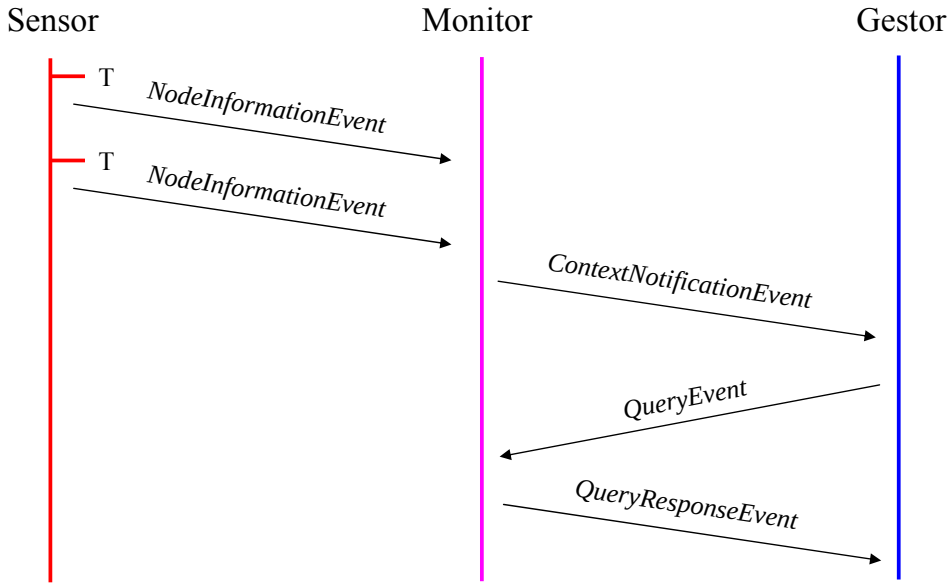

Figura 5.5: Dinâmica de eventos entre os sensores, monitor de contexto e gestor da adaptação

O monitor de contexto recebe eventos *NodeInformationEvent* periodicamente, com informação acerca dos nós. Quando a informação recebida denota uma alteração, é lançado o evento de contexto correspondente, no caso ilustrado na figura, é lançado um *ContextNotificationEvent*. Ocasionalmente o gestor de adaptação

faz interrogações ao repositório de informação do monitor. Estas interrogações são descritas na Secção 5.2.2.1

5.2.2 Componentes de Adaptação

Os componentes de adaptação englobam o gestor da adaptação e os agentes de reconfiguração, sendo responsáveis pelo processo de adaptação, desde a decisão até à aplicação das reconfigurações necessárias. O gestor de adaptação é responsável por determinar as alterações necessárias e coordenar o processo de adaptação. Os agentes de reconfiguração estão distribuídos pelos nós, tal como os sensores de contexto, sendo locais. Estes agentes aplicam as alterações necessárias de acordo com a estratégia para aplicar a reconfiguração estipulada. A Figura 5.6 ilustra os componentes de adaptação, descritos de seguida.

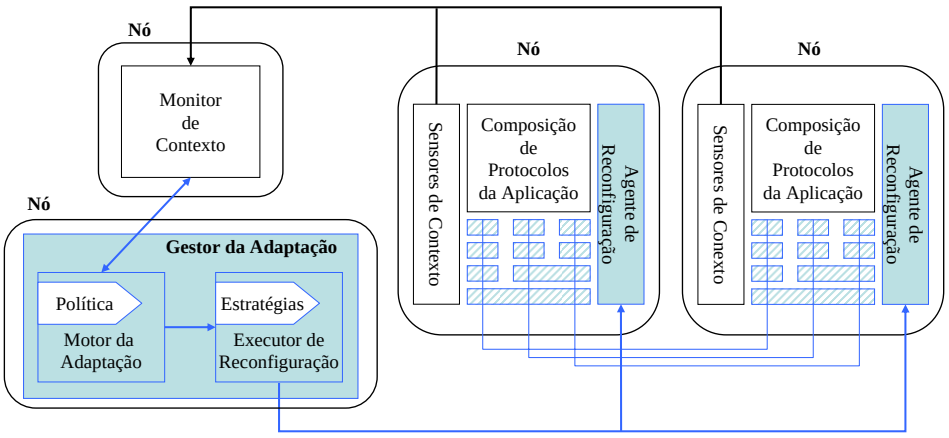

Figura 5.6: Componentes de Adaptação

5.2.2.1 Gestor da Adaptação

O gestor da adaptação é um componente centralizado num nó, responsável pela interpretação da política e pelo controlo da reconfiguração. Este componente é concretizado sob a forma de um protocolo, cuja instância está localizada no mesmo nó que o monitor de contexto. O gestor é constituído por um motor da

adaptação, que realiza a interpretação das políticas, e o executor da reconfiguração, que concretiza as estratégias para aplicar a reconfiguração. Estes componentes são descritos de seguida.

Motor da Adaptação O motor da adaptação avalia a política de adaptação, recebendo os eventos do monitor de contexto, avaliando a política e realizando as interrogações necessárias. O motor da adaptação apenas interage com o monitor de contexto, do qual recebe eventos de contexto e realiza interrogações sobre informação armazenada no repositório.

A avaliação da política de adaptação depende da recepção de eventos de contexto. Os eventos de contexto activam regras especificadas na política. As regras são avaliadas sequencialmente. Sempre que uma regra é despoletada e existam condições (*With*) são efectuadas interrogações ao monitor de contexto. A Figura 5.7 ilustra a dinâmica de eventos para realizar as interrogações.

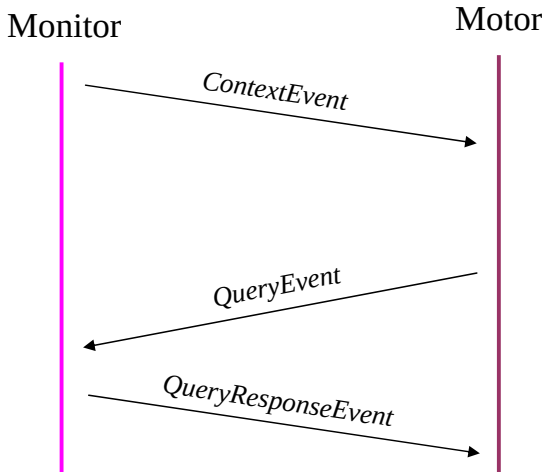

Figura 5.7: Pedido-Resposta de informação de contexto

O motor de adaptação gera um evento *QueryEvent*, que indica qual a informação necessária. O monitor de contexto responde à interrogação com um evento *QueryResponseEvent*, que carrega a informação pedida. Com esta informação o motor pode continuar a avaliar a regra e a política.

O resultado da avaliação da política determina se é necessário efectuar alterações. As alterações necessárias são comunicadas ao executor da reconfiguração, descrito de seguida.

Executor da Reconfiguração O executor da reconfiguração controla o processo de reconfiguração, utilizando um conjunto de directivas aceites pelos agentes. As directivas utilizadas e a sua sequência é definida pela estratégia para aplicar a reconfiguração.

O motor da adaptação, quando a avaliação da política resulta numa acção de reconfiguração, comunica-a ao executor. O executor inicia a aplicação da estratégia definida para a acção recebida. As directivas a enviar ao agente são encapsuladas em eventos. E as respostas do agente à directiva enviada estão também num evento. As estratégias para aplicar a reconfiguração serão descrita em pormenor na Secção 5.2.3.

5.2.2.2 Agentes de Reconfiguração

Em cada nó existe um agente de reconfiguração, responsável por efectuar as alterações necessárias. O agente de reconfiguração tem a capacidade de controlar o fluxo de mensagens, fazer a manutenção do estado dos canais e alterar tanto parâmetros de sessões, como a composição de canais.

Os agentes são concretizados como um protocolo *Appia*, que não afecta a qualidade do serviço do nó. Este protocolo está presente num canal próprio para o controlo da adaptação, juntamente com os sensores presentes no nó. A instância deste protocolo é partilhada por todos os canais que podem ser reconfigurados. Os agentes aceitam um conjunto de eventos que denotam directivas e produzem vários eventos de resposta a directivas, recolha de estado e alteração de parâmetros. Toda esta troca de eventos é realizada com o executor da reconfiguração. Os agentes não interagem entre eles nem com outros componentes da arquitectura. Esta exclusão aplica-se também aos sensores presentes no nó.

A interacção agente-executor é determinada pelas estratégias para aplicar a reconfiguração. Todos os agentes aceitam um conjunto de directivas, que estão

preparados para cumprir. As directivas são realizadas como eventos que implementam a interface *Java* denominada *CoordinationEvent*. Os eventos aceites são os seguintes:

- *PrepareMngAgEvent*: o agente de reconfiguração, perante este evento, prepara as sessões e os canais para a reconfiguração. Esta preparação passa por atingir o estado quiescente, recolher o estado das sessões e interromper o fluxo de mensagens. Estes aspectos são conseguidos através de dois eventos: *PrepareEvent* e *StateEvent*, abordados mais à frente;

- *ActionToPerfMngAgEvent*: o agente recebe neste evento a acção de reconfiguração a concretizar. O evento carrega a informação necessária para a aplicação da acção, como os âmbitos de protocolos e de canais, e informação referente ao processo de reconfiguração. Esta informação indica se é necessário esperar por uma directiva para retomar o envio de mensagens ou não. Após a aplicação da directiva, o estado é actualizado com o recolhido pelo evento *StateEvent*. Após a recuperação do estado, as sessões não começam a processar eventos. O início do processamento de eventos depende não só da informação presente no evento mas também do tipo de acção de reconfiguração. Se tiver sido uma alteração de parâmetros, a continuidade é dada através de um evento *GoEvent*. Se a reconfiguração implicar a renovação do canal, então é utilizada a invocação *Channel.start()*;

- *ContinueMngAgEvent*: o agente perante este evento, dá continuidade ao processamento de eventos através de um *GoEvent* ou invocando o método *Channel.start()*. A escolha depende da acção aplicada.

Para estas directivas existe um conjunto de respostas sob a forma de eventos. Estes eventos também implementam a interface *CoordinationEvent* e são os seguintes:

- *PrepareOKAgMngEvent*: este evento é a resposta à directiva do evento *PrepareMngAgEvent*. É enviado após a recolha do estado através do evento *StateEvent*;

- *ActionDoneAgMngEvent*: este evento é enviado após a aplicação da acção recebida no e recuperação do estado nas sessões através do evento *stateEvent*. Este evento é a resposta à directiva do *ActionToPerfMngAgEvent*;

- *ContinueOKAgMngEvent*: este evento confirma a continuação do processamento de mensagens pelas sessões. Segue-se a um *GoEvent* ou a um *Channel.start()*, quando foi recebido um *ContinueMngAgEvent*.

O agente utiliza vários eventos locais para controlar as sessões dos canais. Estes eventos permitem concretizar localmente as directivas ou, no caso da alteração de parâmetros, aplicar a acção de reconfiguração. Estes eventos são descritos de seguida.

- *PrepareEvent*: este evento obriga as sessões a processarem os eventos que possam ter pendentes e a pararem o envio de eventos;

- *StateEvent*: este evento recolhe o estado das sessões bloqueando-as no processamento de eventos;

- *GoEvent*: este evento desbloqueia as sessões, permitindo a continuação do processamento de eventos. Só é utilizado se precedido por um *SetValueEvent*;

- *SetValueEvent*: este evento, ao contrário dos anteriores, concretiza uma acção de reconfiguração. Este evento é enviado para os canais afectados pela alteração de parâmetros e carrega informação sobre o tipo concreto ou abstractos das sessões alvo, o nome do parâmetro e o novo valor para o parâmetro. Os protocolos reconfiguráveis aceitam este evento.

Dos eventos descritos, os eventos *PrepareEvent* e *StateEvent* são utilizados na técnica de alteração dinâmica da composição dos canais. Como já foi abordado na Secção 5.1.3, esta técnica permite modificar a a pilha de protocolos associada a cada canal, criando um novo canal com a composição desejada. Esta abordagem acarreta várias considerações a ter na criação de um novo canal: a interrupção do fluxo de mensagens e a manutenção do estado. A primeira consideração está a cargo do evento *PrepareEvent*, que obriga as sessões a processarem eventos

pendentes e a não enviarem novos eventos. A segunda consideração é da responsabilidade do evento *StateEvent*, que recolhe a informação de estado das sessões antes da aplicação da acção (que assim ficam bloqueadas no processamento de eventos) e o mesmo evento é responsável por actualizar o estado das novas sessões do novo canal. A Figura 5.8 ilustra o evento de recolha e recuperação do estado. O evento percorre a pilha no sentido ascendente. Quando chega ao topo, o evento é capturado e mantido para posterior recuperação do estado. Após a aplicação da acção de reconfiguração, o mesmo evento de estado é utilizado. O evento percorre a pilha no estado descendente. Cada sessão procura informação de estado no evento, de acordo com o seu tipo de protocolo. Nas situações em que são adicionados protocolos, não existe estado disponível para as novas sessão

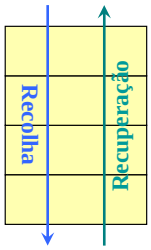

Figura 5.8: Recolha e recuperação do estado do canal

A técnica de alteração dinâmica da composição de um canal utilizada implica que estas considerações sejam tidas ou de forma explícita, no caso das estratégias globais, ou de forma implícita, para as estratégias locais. No primeiro caso, a preparação para a interrupção, a recolha do estado e a paragem do processamento de eventos tem de ser coordenada para garantir que os nós afectados interrompem simultâneamente a produção de evento e que o estado recolhido é coerente nos diversos nós. Por exemplo, o estado de um nó que processa mais uma mensagem é diferente do estado do nó que não a processa, dado que o último número de sequência é diferente para as duas instâncias do mesmo protocolo. Assim, cabe ao executor coordenar as diferentes fases da reconfiguração. No segundo caso, das estratégias locais, as preocupações com o estado e interrupção são tidas localmente, logo estão implícitas numa única directiva, a de efectuar uma reconfiguração, representada pelo evento *ActionToPerfMngAgEvent*.

Os eventos que representam directivas e respostas permitem definir várias estratégias para aplicar a reconfiguração. A definição tanto de estratégias locais como globais é possível com base neste conjunto de eventos de coordenação. A dinâmica de eventos trocada e as próprias estratégias são abordadas na próxima secção.

5.2.3 Estratégias para Aplicar a Reconfiguração

As estratégias determinam como são aplicadas as acções de reconfiguração. As estratégias não são desenvolvidas em função das acções de reconfiguração. Assim uma estratégia pode ser utilizada para todas as acções.

De acordo com as acções de reconfiguração apresentadas neste trabalho, existe a necessidade de estratégias diferentes, com e sem coordenação, pelo que foram desenvolvidas estratégias locais e globais, não tolerantes a falhas.

As estratégias desenvolvidas definem uma sequência de directivas aceites pelos agentes de reconfiguração. O conjunto de directivas utilizadas difere nas estratégias globais e locais e é descrito de seguida.

5.2.3.1 Estratégia Local

A estratégia local concretizada, aplica a reconfiguração de forma independente dos outros nós e tem em consideração a manutenção do estado, bem como a confirmação da aplicação da acção no nó. A estratégia utiliza apenas uma directiva representada pelo evento *ActionToPerfMngAgEvent*. Este evento indica ao agente qual a acção de reconfiguração a realizar e, no caso desta estratégia, que pode dar continuação ao processamento de eventos imediatamente após a conclusão da acção. O agente utiliza os eventos de *PrepareEvent* e *StateEvent* para a recolha do estado, antes de aplicar a reconfiguração. Após a aplicação da reconfiguração, o agente recupera o estado através do evento *StateEvent* e continua o processamento ou através de um *GoEvent* (no caso de uma acção *SetValue*) ou através de uma invocação *Channel.start()* (no caso das restantes acções, dado que utilizam a técnica de alteração dinâmica). Após este último passo, o agente de

reconfiguração confirma a conclusão da reconfiguração através de um evento *ActionDoneAgMngEvent*. A dinâmica de eventos associada à estratégia local descrita é ilustrada na Figura 5.9.

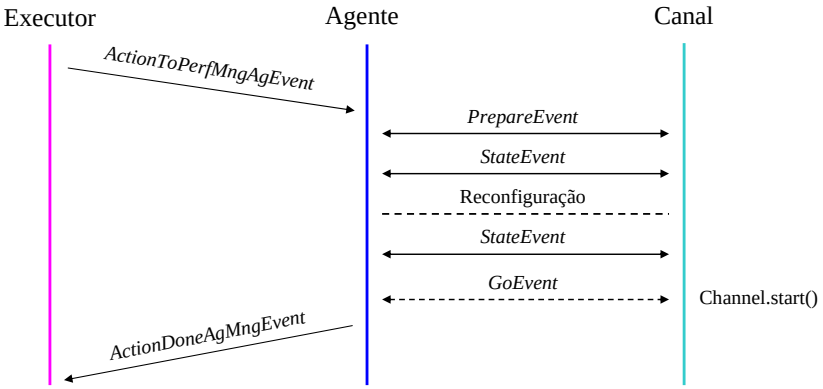

Figura 5.9: Dinâmica de eventos na estratégia local

5.2.3.2 Estratégia Global

A estratégia global oferece garantias de coordenação entre os nós do sistema, controlando explicitamente a manutenção do estado e as restantes fases do processo de reconfiguração. A estratégia utiliza todas as directivas definidas na Secção 5.2.2.2.

A primeira directiva enviada pelo executar é representada pelo evento *PrepareMngAgEvent* com o objectivo de preparar os alvos da reconfiguração antes da aplicação da acção. Na recepção do evento o agente procede ao envio dos eventos *PrepareEvent* e posteriormente *StateEvent*, garantindo que os nós atingem um estado quiescente e o estado das sessões é recolhido. Após o cumprimento da directiva, o agente confirma a conclusão ao executor através de um evento *PrepareOKAgMngEvent*. No momento em que todos os nós confirmaram, o executor tem a garantia de que nenhum nó continua a produção ou processamento de eventos e o estado é coerente entre eles. O executor avança para a fase seguinte do processo e envia a segunda directiva através do evento *ActionToPerfEvent* para todos os nós. Este evento carrega a acção de reconfiguração e, no caso desta

estratégia, a ordem para aguardar por uma directiva para continuar o processamento. Os agentes efectuam a reconfiguração e recuperam o estado através do mesmo evento *StateEvent*. Após a conclusão da segunda directiva, o agente envia a confirmação através do evento *ActionDoneAgMngEvent*. Após a confirmação de todos os nós, já preparados para continuar a execução, o executor inicia a última fase. A directiva representada pelo evento *ContinueMngAgEvent* indica aos agentes para desbloquearem as sessões ou iniciarem os canais novos. No primeiro caso, o agente envia um evento *GoEvent*. No segundo caso, é invocado o método *Channel.start()*. No fim, o agente confirma ao executor através de um *ContinueOKAgMngEvent* que conclui a troca de eventos. A dinâmica de eventos descrita é ilustrada pela Figura 5.10

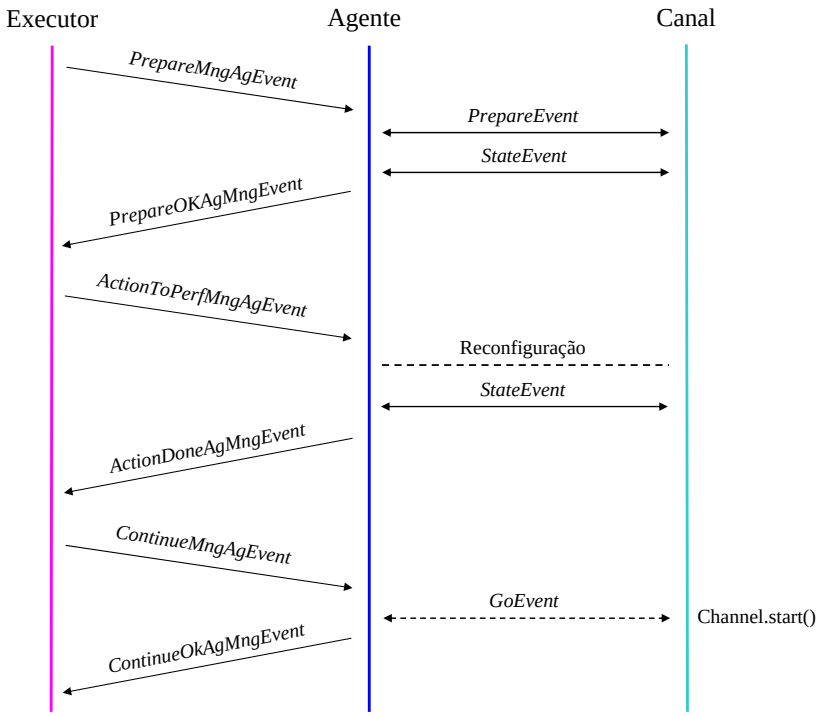

Figura 5.10: Dinâmica de eventos na estratégia global

Sumário

Neste capítulo foi apresentada a concretização da arquitectura descrita no Capítulo 3, abordando em pormenor os componentes de contexto e de adaptação. A avaliação do protótipo construído é realizada com base num caso de estudo, descrito no capítulo seguinte.

Capítulo 6

Avaliação

Neste capítulo é apresentado o caso de estudo construído sobre a concretização da arquitectura descrita no capítulo anterior. A aplicação de demonstração realizada, os protocolos utilizados, o modelo de contexto definido, a política de adaptação especificada e as estratégias utilizadas para cada acção de reconfiguração são descritas nas secções que se seguem. Por fim, são apresentados os resultados relativos à duração do processo de reconfiguração para diferentes acções e número de nós afectados.

6.1 Caso de Estudo

O caso de estudo consiste numa aplicação para a observação do movimento de uma bola que se desloca num espaço finito. A bola é observada por vários elementos mas o seu movimento é alterado apenas por um elemento, responsável por informar os restantes da posição da bola.

A bola movimenta-se da seguinte forma. Durante um período de duração aleatória, a bola desloca-se com direcção e velocidade constantes. O período termina quando é escolhida uma nova direcção e o elemento é repetido. A perturbação introduzida no movimento de cada período é aleatória, pelo que cada período de tempo apresentará movimentos diferentes.

Existem diferentes composições de protocolos que podem ser utilizados para efectuar a propagação do estado da bola e observação do seu movimento. A adaptação, para este caso de estudo, consiste em seleccionar e instalar a composição

mais adequada, em função das características do movimento da bola e do estado da rede.

6.1.1 Observação do Movimento da Bola

A posição da bola é transmitida periodicamente pelo elemento modificador, que tem capacidade de alterar o movimento. Cabe também a este elemento efectuar as alterações e, durante as fases de alteração, transmitir a posição da bola. Os elementos observadores limitam-se a posicionar a bola de acordo com as indicações recebidas do elemento modificador.

6.1.2 Alteração do Movimento da Bola

A bola tem uma posição e velocidade iniciais. A posição da bola é dada pelas coordenada x e y. A velocidade da bola é dada por um vector. O elemento modificador do movimento da bola introduz um factor aleatório que altera o movimento. O factor aleatório afecta a direcção do movimento, não afectando o módulo da velocidade, que se mantém ao longo do tempo. O factor aleatório introduzido é um ângulo em radianos, que é aplicado ao vector da velocidade, através das funções seno e coseno.

A introdução de um factor aleatório para alterar o movimento permite distinguir dois tipos de movimento: previsível, até à aplicação do factor aleatório, e imprevisível quando o factor é aplicado. A aplicação do factor aleatório e o seu valor é determinado pelo elemento modificador.

6.1.3 Predição do Movimento da Bola

Enquanto o movimento da bola for previsível, é possível, através de um protocolo de predição preparado para o efeito, determinar qual a próxima posição da bola. Isto deve-se ao facto do vector da velocidade se manter inalterada. No entanto, sempre que é aplicado o factor aleatório, o desconhecimento deste valor, resulta numa falha da previsão da próxima posição da bola.

A previsão está sujeita a uma métrica de avaliação, que compara, ao longo do tempo a posição real da bola (informação fornecida pelo elemento modificador)

e a posição prevista pelo protocolo. Esta métrica reflecte-se como uma taxa de erros de previsão.

6.2 Protocolos

O caso de estudo introduzido nas secções anteriores utiliza diferentes configurações de protocolos de acordo com as características do movimento da bola e do estado da rede. São utilizados vários protocolos nessas configurações. A Figura 6.1 ilustra a hierarquia de tipos de protocolos construída para o caso de estudo. No caso dos canais, não foi necessário estender a hierarquia. Estes protocolos são descritos nas secções que se seguem e posteriormente são abordadas as várias configurações utilizadas.

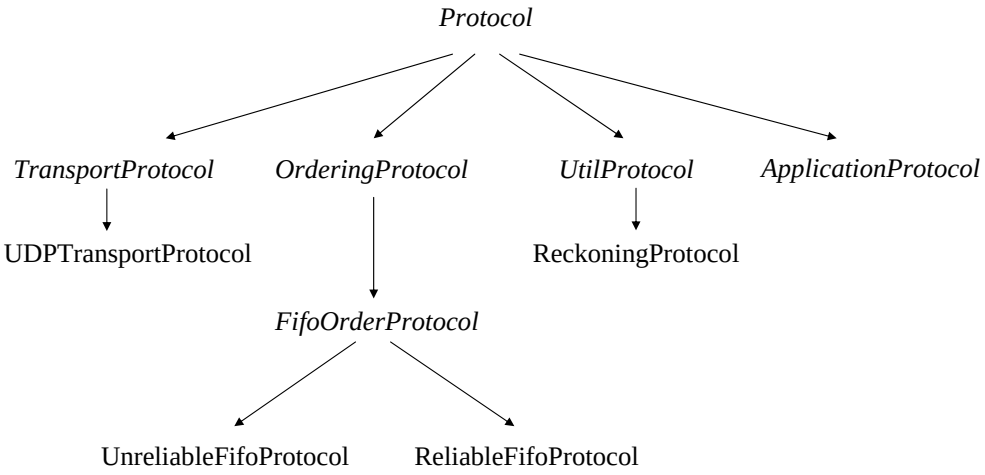

Figura 6.1: Hierarquia de tipos de protocolos para o caso de estudo

6.2.1 Protocolo da Aplicação

A aplicação de demonstração desenvolvida para este caso de estudo é concretizada sob a forma de um protocolo *RAppia*. Este protocolo não afecta a qualidade de serviço.

O protocolo da aplicação lança a interface gráfica que permite visualizar o movimento da bola. Este protocolo é responsável ou por controlar o movimento

da bola e propagá-lo para os nós que o observam, ou por reproduzir o movimento de acordo com as posições propagadas pelo elemento modificador.

O protocolo não fornece informação de contexto. No entanto, a informação de contexto do nó é apresentada ao utilizador, por meio da recepção do evento *NodeInformation*, que fornece a última informação disponível das várias informações de contexto. Para além de não disponibilizar informação de contexto, o protocolo não possui parâmetros reconfiguráveis por elementos externos, ou seja, pelo agente de reconfiguração. Contudo, a interface gráfica da aplicação permite alterar o parâmetro reconfigurável do protocolo de descarte (abordado na Secção 6.2.4). A reconfiguração é conseguida através de um evento *SetDiscardValueEvent*. A aplicação, apesar de apresentar diversa informação que pode ser considerada estado, não a trata como tal, dado que a sua execução não é interrompida.

O protocolo da aplicação processa o evento *BallPositionEvent*. Este evento contém a posição da bola, bem como informação acerca da sua direcção e velocidade do movimento da bola. O elemento modificador da bola é responsável por gerar este evento, periodicamente, para que seja possível aos outros elementos reproduzirem o movimento. A Tabela 6.1 resume as características do protocolo.

		Application
Tipo		ApplicationProtocol
Estado		
Contexto		
Parâmetros		
Eventos	Reconfiguração	SetDiscardValueEvent
	Gerais	BallPositionEvent NodeInformationEvent
	Captura Ctx	

Tabela 6.1: Características do protocolo da aplicação

6.2.2 Protocolo de Predição

O protocolo de predição (*Reckoning*) é um protocolo que prevê o movimento da bola, com base na sua última posição, velocidade e direcção. Foi desenvolvido

concretamente para o caso de estudo, implementando a interface do tipo concreto *ReckoningProtocol*. O protocolo oferece posições estimadas da bola, utilizadas sempre que a posição da bola original não é recebida.

O protocolo de predição não só fornece a próxima posição da bola, como também disponibiliza informação de contexto relativa ao seu funcionamento. A informação de contexto refere-se à taxa de erros na previsão do movimento. Sempre que o protocolo recebe um evento *BallPositionEvent*, compara a posição real com a estimada, calculando uma taxa de erros.

O protocolo não é reconfigurável a nível de parâmetros mas tem estado. O estado resume-se à última posição da bola recebida, na qual baseia a predição. Assim, este protocolo aceita o evento de estado, deposita a informação e também a recupera. A Tabela 6.2 resume as características do protocolo.

		Reckoning
Tipo		ReckoningProtocol
Estado		
Contexto		ReckoningErrorRate
Parâmetros		
Eventos	Reconfiguração	
	Gerais	BallPositionEvent
	Captura Ctx	GetValueEvent ReturnValueEvent

Tabela 6.2: Características do protocolo de predição

6.2.3 Protocolos de Ordem *FIFO*

Os protocolos de ordem *FIFO* (*first in, first out*) garantem que as mensagens são ordenadas pela sua sequência no tempo, ou seja, se uma mensagem a foi enviada antes de b então a é entregue antes de b. A ordenação baseia-se na atribuição de números de sequência a cada uma das mensagens. A ordenação FIFO na entrega das mensagens é uma garantia suficiente para assegurar a coerência nos aspectos da comunicação no caso de estudo descrito. São utilizados dois protocolos de ordem *FIFO*, descritos de seguida.

6.2.3.1 *Unreliable FIFO*

O protocolo *UnreliableFIFOProtocol* (UFIFO), do tipo concreto com o mesmo nome, não oferece garantias de fiabilidade na entrega de mensagens, pelo que as mensagens perdidas não são retransmitidas. O protocolo possui informação de estado, que se refere ao último número de sequência atribuído. Assim, o protocolo aceita o evento de estado, depositando ou recuperando a informação relativa ao número de sequência.

O protocolo UFIFO disponibiliza informação de contexto: a taxa de omissões, ou seja, de pacotes perdidos. A captura desta informação ocorre sempre que é recebido um evento *GetValueEvent* com o campo *OMISSIONRATE*. A disponibilização da informação passa por responder com um evento *ReturnValueEvent* com a especificação do tipo de taxa (*OMISSIONRATE*) e o seu valor.

Este protocolo não é reconfigurável, ignorando os eventos *SetValueEvent*, responsáveis pela reconfiguração. Em contrapartida, todos os eventos que estendem a classe *SendableEvent* são aceites e é-lhes atribuído um número de sequência. A classe *SendableEvent* identifica os eventos que são enviados para outros nós. A Tabela 6.3 resume as características do protocolo.

		UFIFO
Tipo		UnreliableFifoProtocol
Estado		Nº de sequência
Contexto		OmissionRate
Parâmetros		
Eventos	Reconfiguração	
	Gerais	SendableEvent
	Captura Ctx	GetValueEvent ReturnValueEvent

Tabela 6.3: Características do protocolo de ordem FIFO não fiável

6.2.3.2 *Reliable FIFO*

O protocolo *ReliableFIFOProtocol* (RFIFO), que implementa a interface do tipo com o mesmo nome, oferece fiabilidade na entrega das mensagens, ou seja, se uma mensagem for perdida, ela é retransmitida. Tal como o protocolo UFIFO,

este protocolo possui como informação de estado o último número de sequência atribuído.

O protocolo RFIFO disponibiliza informação de contexto, relativa à taxa de retransmissões, ou seja, o número de mensagens que voltaram a ser transmitidas porque não foram recebidas. A captura desta informação ocorre sempre que é recebido um evento *GetValueEvent* com o campo *RETRANSRATE*. O protocolo responde com um evento *ReturnValueEvent*, com a especificação do tipo de taxa (*RETRANSRATE*) e o seu valor.

Ao contrário do protocolo UFIFO, o RFIFO é reconfigurável a nível de parâmetros. O parâmetro *timeoutValue*, que determina o tempo de espera antes de retransmitir uma mensagem, pode ser reconfigurado. A reconfiguração ocorre sempre que é recebido um evento *SetValueEvent* com o nome do parâmetro referido e o novo valor.

Para além dos eventos já referidos, todos os eventos que estendem a classe *SendableEvent* são aceites. Esta classe identifica os eventos que são enviados para outros nós e são estes os eventos que são ordenados pelo protocolo. A Tabela 6.4 resume as características do protocolo.

		RFIFO
Tipo		ReliableFifoProtocol
Estado		Nº de sequência
Contexto		RetransmissionRate
Parâmetros		timeoutValue
Eventos	Reconfiguração	SetValueEvent
	Gerais	SendableEvent
	Captura Ctx	GetValueEvent ReturnValueEvent

Tabela 6.4: Características do protocolo de ordem FIFO fiável

6.2.4 Protocolo de Descarte

Este protocolo simula a perda de mensagens. Em média, a cada x mensagens, uma é descartada, não sendo entregue. A perda de mensagens não ocorre em períodos exactamente iguais, é introduzido um factor aleatório. Este protocolo

denomina-se *DiscardProtocol* e implementa a interface do tipo concreto com o mesmo nome.

O parâmetro *discardValue*, referente ao valor x, é reconfigurável, não do ponto de vista do agente, mas sim do utilizador que controla a aplicação de demonstração, dado que é possível introduzir um novo valor na interface gráfica. O novo valor é encapsulado num evento *SetDiscardValue* produzido pelo protocolo da aplicação. O parâmetro x constitui ainda a informação de estado a manter.

Para além do evento referido anteriormente, o protocolo aceita eventos *BallPositionEvent*. Estes eventos podem ser descartados pelo protocolo, com o objectivo de variar a taxa de omissões no nó. O protocolo não interfere com outros eventos, nem fornece informação de contexto. A Tabela 6.5 resume as características do protocolo.

		Discard
Tipo		DiscardProtocol
Estado		discardValue
Contexto		
Parâmetros		discardValue
Eventos	Reconfiguração	SetDiscardValueEvent
	Gerais	BallPositionEvent
	Captura Ctx	

Tabela 6.5: Características do protocolo de descarte

6.2.5 Protocolo UDP

O protocolo UDP tem o tipo *UDPTransportProtocol* e permite enviar mensagens entre os nós. Não garante fiabilidade e é sensível às condições da rede. Este protocolo aceita todos os eventos que estendam a classe *SendableEvent* e envia-os para os seus destinos. O protocolo UDP não apresenta estado nem informação de contexto relevante para o caso de estudo. A Tabela 6.6 resume as características do protocolo.

		UDP
Tipo		UDPTransportProtocol
Estado		
Contexto		
Parâmetros		
Eventos	Reconfiguração	
	Gerais	SendableEvent
	Captura Ctx	

Tabela 6.6: Características do protocolo UDP

6.2.6 Configurações da Pilha de Protocolos

O caso de estudo pressupõe que existe uma distribuição dos elementos observadores e modificador em nós diferentes. Cada nó corre a aplicação, sobre uma pilha de protocolos *RAppia*. A pilha de protocolos é adaptada em função das condições da rede bem como do retorno dado pelo protocolo de predição. As condições da rede são caracterizadas pela taxa omissões e pela taxa de retransmissões. O protocolo de predição fornece uma taxa de erros que reflecte a sua exactidão.

A pilha de protocolos pode ter várias configurações. Cada configuração é orientada a uma situação específica mas todas as configurações resultam em canais do mesmo tipo: *DataChannel*. As três configurações possíveis para o caso de estudo abordado são ilustradas pela Figura 6.2. Todas as configurações apresentam a aplicação (*Aplication*), o protocolo de transporte escolhido (*UDP*) e um protocolo de ordenação de mensagens (*UFIFO* ou *RFIFO*). O protocolo de predição (*Reckoning*) está presente apenas em alguma situações.

As configurações ilustradas reflectem três qualidades de serviço distintas. A primeira configuração apresentada na figura é a configuração inicial de cada nó. Esta configuração é a mais simples, consumindo menores recursos e com o desempenho mais rápido. No entanto, dado que apresenta o protocolo *UFIFO*, esta configuração não oferece garantias de fiabilidade de ordenação e entrega das mensagens.

A segunda configuração apresenta um protocolo de predição, para além dos protocolos *UFIFO* e *UDP*. Esta configuração é orientada às situações em que a rede apresenta uma latência elevada e, consequentemente, a taxa de omissões

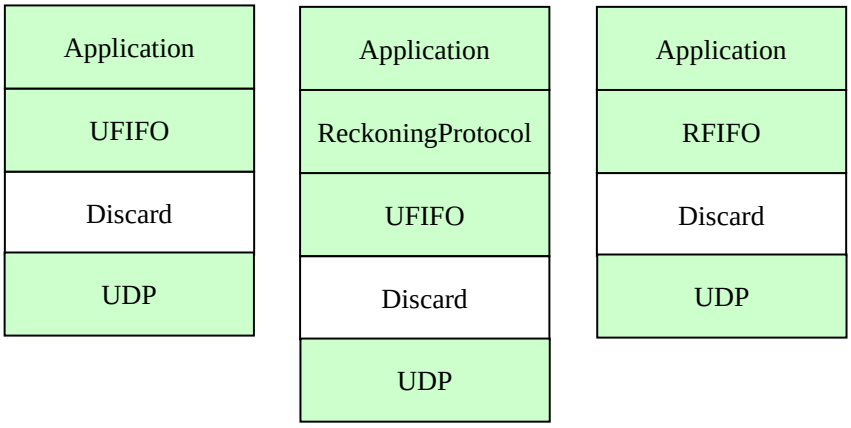

Figura 6.2: Configurações possíveis da pilha de protocolos

aumenta. A utilização do protocolo de predição permite fazer uma previsão das próximas posições da bola e utilizá-las sempre que as posições reais não estejam disponíveis.

A terceira e última configuração não apresenta o protocolo de predição mas, em compensação, tem o protocolo de ordenação *RFIFO*. Esta configuração oferece fiabilidade na ordenação e entrega das mensagens, sendo portanto, mais pesada a nível de consumo de recursos. Assim, a configuração é orientada a situações em as condições da rede se encontram muito deterioradas, mantendo a fluidez do movimento da bola.

Todas as configurações apresentam o protocolo de descarte (*Discard*), representado na cor branca, dado que não afecta a qualidade de serviço. A presença protocolo, que permite obter a taxa de omissões, simula as condições da rede, relativamente à perda de mensagens.

Cada nó tem apenas uma configuração de cada vez. As configurações não têm de ser iguais em todos os nós, para que haja compatibilidade. A Tabela 6.7 ilustra quais as configurações que podem existir simultaneamente.

As qualidades de serviço desejadas, para situações distintas, definem a lógica de adaptação. O modelo de contexto e a política de adaptação reflectem as necessidades adaptativas do caso de estudo. Estes elementos são abordados nas secções seguintes.

	UDP+UFIFO	UDP+UFIFO+RECK	UDP+RFIFO
UDP+UFIFO	x	x	
UDP+UFIFO+RECK	x	x	
UDP+RFIFO			x

Tabela 6.7: Compatibilidade das configurações

6.3 Modelo de Contexto

A informação de contexto e os eventos gerados a partir da monitorização dependem da aplicação. Assim, para o caso de estudo descrito, foram definidas as informações observáveis e interpretadas necessárias, bem como os eventos de notificação do contexto e em que situações são lançados.

6.3.1 Informações Observáveis e Interpretadas

As informações de contexto necessárias para o caso de estudo prendem-se com as características da rede e do movimento da bola. Assim, foram definidas várias informações observáveis e derivadas que permitem monitorizar esses aspectos.

As informações são definidas com base num conjunto de tipos já existentes: *double*, *bool* e *nodeId*. Este conjunto foi estendido, passando a integrar o tipo *protocolType*, que define um tipo de protocolo abstracto ou concreto. As informações especificadas são as seguintes:

```
contextType NodeInfo
        sensed observable retransmissionRate: nodeId -> double
        sensed observable omissionRate: nodeId -> double
        sensed observable reckoningErrorRate: nodeId -> double
        derived observable regularReckER : nodeId -> bool
         regularOmissionR(nodeId) = (hasProtocol(nodeId,UFIFO) &&
             omissionRate(NodeId)< MAXOMISSIONRATE )
        derived observable regularReckER : nodeId -> bool
         regularReckER(nodeId) = (hasProtocol(nodeId,Reckoning) &&
             reckoningErrorRate(NodeId)< MAXERRORDOWN )
        derived observable regularRetransR : nodeId -> bool
         regularRetransR(nodeId) = (hasProtocol(nodeId,RFIFO) &&
             retransmissionRate(NodeId)< MAXRETRANS )
```

```
contextType NodeConf
        sensed observable hasProtocol: nodeId protocolType -> bool
```

As informações definidas organizam-se em dois tipos de contexto *NodeInfo* e *NodeConf*. O primeiro tipo engloba várias informações observáveis (*sensed*) e derivadas (*derived*) em função do nó, ao passo que o segundo apenas tem uma informação observável. O tipo *NodeInfo* tem as seguintes informações de contexto:

- *retransmissionRate*: a taxa de retransmissões é uma informação observável relativa ao protocolo *RFIFO*, denotando o número de mensagens retransmitidas. Esta informação é acedida através da identificação do nó (*nodeId*), sendo devolvido um valor *double*;

- *omissionRate*: a taxa de omissões é uma informação observável relativa ao número de mensagens perdidas pelo protocolo *UFIFO*. Esta informação, tal como a anterior, é acedida através da identificação do nó e é devolvido um valor;

- *reckoningErrorRate*: a taxa de erros do protocolo de predição (*Reckoning*) é uma informação observável, que denota a percentagem de previsões bem sucedidas. Esta informação é acedida da mesma forma que as anteriores;

- *regularReckER*: esta informação derivada denota um nó que tenha o protocolo de tipo *Reckoning* e cuja taxa de erros (*reckoningErrorRate*) seja inferior a MAXERRORDOWN. A informação é acedida através do *nodeId* e devolve um booleano;

- *regularOmissionR*: esta informação derivada denota um nó que tenha o protocolo de tipo *UFIFO* e cuja taxa de omissões (*omissionRate*) seja inferior a MAXOMISSIONRATE. A informação é acedida da mesma forma que a anterior;

- *regularRetransR*: esta informação derivada denota um nó que tenha o protocolo de tipo *RFIFO* e cuja taxa de retransmissões (*retransmissionRate*) seja inferior a MAXRETRANS. A informação é acedida da mesma forma que as anteriores.

O tipo *NodeInfo* tem apenas uma informação observável *hasProtocol*, que indica se um tipo de protocolo (*protocolType*) está presente num nó. A informação é acedida através da identificação do nó e do tipo de protocolo , devolvendo um booleano.

6.3.2 Eventos de Notificação do Contexto

As informações de contexto definidas na secção anterior permitem monitorizar os aspectos relevantes do contexto do sistema. Da monitorização resulta a detecção de alterações relevantes. Estas alterações são notificadas sobre a forma de eventos de contexto. Os eventos necessários são definidos de seguida.

```
event RetransmissionRateEvent
        parameter nodeId : nodeId
        parameter value: double

event OmissionRateEvent
        parameter nodeId : nodeId
        parameter value : double

event ReckoningErrorRateEvent
        parameter nodeId : nodeId
        parameter value : double

event SingleRegularConditionsEvent
        parameter nodeId : nodeId

event GlobalRegularConditionsEvent
```

Os eventos definidos estão estritamente relacionados com as informações observáveis e derivadas, definidas na secção anterior. Os eventos podem ter parâmetros ou não. A descrição de cada evento e a sua geração pelo monitor de contexto é descrita de seguida:

- *RetransmissionRateEvent*: o evento é gerado sempre que existe uma variação significativa da taxa de retransmissão de um nó. Considera-se significativa uma variação de 45%. A alteração é detectada através da monitorização da informação *retransmissionRate*. O evento transporta informação acerca

do nó (*nodeId*) a que a alteração está associada e qual o novo valor da taxa (*value*).

- *OmissionRateEvent*: o evento é gerado quando o número de pacotes perdidos varia de forma significativa, ou seja, quando a taxa de omissões varia em 31%. Esta alteração é detectada através da monitorização da informação *omissionRate*. O evento denota a alteração num nó *nodeId* com o novo valor *value*.

- *ReckoningErrorRateEvent*: este evento, tal como os anteriores, é gerado sempre que existe uma variação significativa da taxa de erros do protocolo de predição, neste caso de 60%. A alteração verifica-se num nó específico e é detectada observando a informação *reckoningErrorRate*. O evento transporta informação acerca do nó (*nodeId*) a que a alteração está associada e qual o novo valor da taxa (*value*).

- *SingleRegularConditionsEvent*: este evento é gerado quando existe uma diminuição na taxa de erros do protocolo de predição, de forma a que esta seja inferior a MAXERRORDOWN, desde que a taxa de omissões também se encontre abaixo do limiar MAXOMISSION. Esta alteração é detectada nos nós que tenham o protocolo de predição (*Reckoning*) na configuração da pilha protocolar, observando as informações *regularReckER* e *regularOmissionR*. O evento transporta informação sobre o nó *nodeId* onde a alteração ocorreu.

- *GlobalRegularConditionsEvent*: este evento é gerado sempre que, em todos os nós do sistema, a taxa de retransmissões diminui, sendo inferior ao limiar MAXRR. Esta alteração é detectada através da monitorização da informação *regularRetransR* em todos os nós. O evento não carrega informação.

Os eventos definidos de acordo com as necessidades de especificação da lógica de adaptação, são utilizados para despoletar as regras da política. A política de adaptação é descrita de seguida.

6.4 Política de Adaptação

As regras da política de adaptação descrevem as transições da configuração da pilha de protocolos consoante o contexto, bem como afinações de parâmetros reconfiguráveis de protocolos. As regras baseiam-se nos eventos definidos e na informação de contexto acessível através de interrogações, para especificarem as reconfigurações necessárias.

A pilha de protocolos atribuída inicialmente a cada nó, não oferece garantias de fiabilidade, logo sendo a mais leve a nível de consumo de recursos. No entanto, quando as condições da rede se deterioram, esta configuração não é suficiente, pelo que são necessárias reconfigurações. A Figura 6.3 ilustra as várias transições de configuração e os eventos que as originam. As secções seguintes descrevem cada uma das transições.

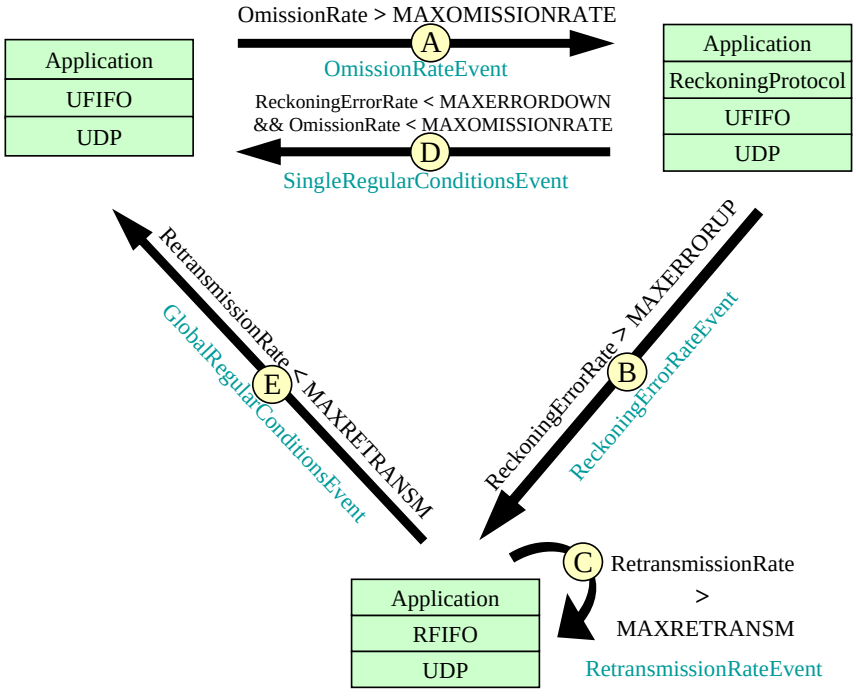

Figura 6.3: Transições especificadas da política

6.4.1 Transição A

Esta transição ocorre perante a deterioração das condições da rede, visível através do aumento da taxa de omissões, fornecida pelo protocolo *UFIFO*. A reconfiguração adequada passa por adicionar um protocolo de predição (*ReckoningProtocol*). Sempre que não é recebida a posição da bola, é utilizada a posição prevista. A reconfiguração é local e é definida pela regra seguinte:

```
WHEN OmissionRateEvent: OmissionRateEvent.value > MAXOMISSION
WITH !NodeInfo.hasProtocol(OmissionRateEvent.nodeId,
    ReckoningProtocol)
DO addProtocol(ReckoningProtocol,above)
        WHERE {OmissionRateEvent.nodeId}
        FOR OrderingProtocol
        APPLY DataChannel
```

6.4.2 Transição B

Após a transição A, o protocolo de predição está em funcionamento, gerando uma taxa de erros relativa à previsão do movimento. Quando a taxa de erros ultrapassa o limiar MAXERRORR, o movimento da bola previsto não é correcto. A reconfiguração adequada passa por substituir a configuração da pilha por uma que ofereça garantias melhores, nomeadamente fiabilidade na ordenação das mensagens. A substituição é realizada globalmente, dado que os protocolos de ordenação não são compatíveis entre si. A regra que dita a substituição da pilha é a seguinte:

```
WHEN ReckoningErrorRateEvent: ReckoningErrorRateEvent.value >
    MAXERRORRUP
DO changeQoS(UDPProtocol,RFIFOProtocol,Application)
        APPLY DataChannel
```

6.4.3 Transição C

Após a transição B, a configuração da pilha apresenta o protocolo de ordenação fiável *RFIFO*. A deterioração das condições da rede é visível através da taxa de

retransmissão de mensagens. Nas situações em que a taxa é superior a MAXRETRANSM pretende-se aumentar o período de tempo de entrega das mensagens para TIMEOUTVALUE, minorando a carga da rede com retransmissões desnecessárias. Caso o valor do parâmetro já seja TIMEOUTVALUE, cabe à sessão alvo, descartar a reconfiguração. A regra seguinte define a reconfiguração descrita, cuja afectação é local.

```
WHEN RetransmissionRateEvent: RetransmissionRateEvent.value >
    MAXRETRANSM
DO setValue(timeoutValue,TIMEOUTVALUE)
        WHERE {RetransmissionRateEvent.nodeId}
        FOR OrderingProtocol and DataChannel
```

6.4.4 Transição D

A transição D ocorre quando as condições da rede regressam aos parâmetros definidos como aceitáveis, a configuração atribuída inicialmente aos nós é a mais adequada, oferecendo um compromisso entre garantias e consumo de recursos. O regresso á configuração inicial depende da actual configuração da pilha. Caso as condições se apresentem normais quando está em funcionamento o protocolo de predição, basta retirá-lo. Apenas é afectado o nó onde as condições normalizaram. A regra que define a reconfiguração é a seguinte:

```
WHEN SingleRegularConditionsEvent
DO removeProtocol()
        WHERE {SingleRegularConditionsEvent.nodeId}
        FOR ReckoningProtocol
        APPLY DataChannel
```

6.4.5 Transição E

A transição E ocorre quando as condições da rede normalizam, num cenário em que é utilizado o protocolo de ordenação fiável *RFIFO*. A reconfiguração adequada consiste em substitui-lo pelo protocolo não fiável *UFIFO*. Esta substituição afecta todos os nós e como tal, só deve ocorrer quando o contexto global reflecte uma estabilização das condições normais.

```
WHEN GlobalRegularConditionsEvent
DO changeProtocol(UFIFOProtocol)
        FOR OrderingProtocol
        APPLY DataChannel
```

Esta transição, tal como as anteriores, é definida por uma regra da política que determina qual a acção de reconfiguração necessária. A aplicação da acção está a cargo de estratégias locais e globais, apresentadas na secção seguinte.

6.5 Acções de Reconfiguração e Estratégias

As estratégias locais e globais concretizadas foram utilizadas para aplicar as várias acções de reconfiguração. Para o caso de estudo descrito neste trabalho foi seleccionada apenas uma estratégia para cada acção, independentemente dos âmbitos topológico, de protocolos e de canais. Assim, a selecção é descrita de seguida:

- *Transição A*: a acção *addProtocol* é aplicada por uma estratégia local e implica a utilização da técnica de alteração dinâmica, dado que afecta a composição do canal. Contudo, visto que a configuração resultante da adição do protocolo de predição, é compatível com a inicial, não é necessário alterar todos os nós.

- *Transição B*: a acção *changeQoS* é aplicada por uma estratégia global que garante que todos os nós estão coordenados e realizam a reconfiguração simultaneamente, garantindo que todos os nós afectados iniciam o processamento de eventos na mesma altura, não existindo incompatibilidades.

- *Transição C*: a acção *setValue* é aplicada por uma estratégia local. A escolha prende-se com o facto de não ser necessária a coordenação entre os nós para afinar os parâmetros dos protocolos.

- *Transição D*: a acção *removeProtocol* é aplicada através de uma estratégia local e utiliza a técnica de alteração dinâmica dos canais. A remoção ocorre apenas para o protocolo de predição, e a configuração resultante é compatível com a configuração inicial, pelo que a estratégia local é suficiente.

- *Transição E*: a acção *changeProtocol*, apesar de lidar apenas com um protocolo, afecta o protocolo de ordenação. Dado que os protocolos não são compatíveis entre si, é necessário reconfigurar todos os nós do sistema, pelo que é necessária uma estratégia global para garantir que todos os nós se reconfiguram simultaneamente e iniciam a processamento e geração de eventos com uma configuração igual.

De acordo com as estratégias definidas para cada uma das transições descritas e a utilização ou não da técnica de alteração dinâmica da composição, é possível efectuar medidas do intervalo de tempo necessário para que a estratégia aplique a reconfiguração. A avaliação destas medidas é realizada na secção seguinte.

6.6 Análise Qualitativa

A fluidez do movimento da bola depende de vários aspectos, como a frequência com que a posição da bola é desenhada, no elemento modificador, ou a frequência com que a posição da bola é recebida, no elemento observador. O elemento modificador depende dos recursos disponíveis e o observador, das condições da rede.

A política de adaptação proposta visa, essencialmente, colmatar a falta de fluidez do movimento nos elementos observadores. A perda de mensagens correspondentes à posição da bola, implica, que no observador, a bola se mantenha na mesma posição. Quando finalmente a posição da bola é recebida, ocorre uma movimentação abrupta da bola. O problema da falta de fluidez é abordado com um protocolo de predição que fornece a posição da bola, quando está em falta. A fluidez do movimento sofre uma melhoria significativa observada em taxas de omissão elevadas, pelo que a adição do protocolo de predição é adequada e verifica uma melhoria.

No entanto, a introdução de um protocolo de predição revela problemas, quando a alteração do movimento da bola, através de um factor aleatório, é frequente. Nestes casos, é observado um movimento incorrecto da bola até ser recebida uma posição da bola, já com a direcção correcta do movimento. Quanto

mais elevada for a taxa de omissão, maior é o intervalo de tempo em que se observa o movimento incorrecto da bola. Quando uma posição correcta é recebida, verifica-se uma alteração abrupta do movimento, aparecendo a bola numa posição bastante distinta. A alteração da configuração do canal, oferecendo entrega fiável de mensagens, observa a retorno à fluidez do movimento. Infelizmente, devido a constrangimentos temporais, não foi possível realizar experiências executando este protótipo sobre dispositivos móveis. Em PCs, comunicando através de uma rede local, o custo adicional da camada em RFIFO em relação à camada UFIFO não é significativo. No entanto, o facto da camada UFIFO não obrigar à troca de confirmações torna óbvio que, nos dispositivos em que a energia disponível é limitada, a utilização da UFIFO é preferível sempre que as condições da rede a permitam, razão que motiva a transição E. Nas experiências efectuadas não se observou uma quebra na fluidez do movimento quanto esta reconfiguração foi aplicada.

6.7 Tempos de Aplicação da Reconfiguração

Embora o ênfase deste trabalho não esteja nas estratégias para aplicar a reconfiguração, é interessante observar qual o tempo necessário para a realização destas, apesar dos resultados obtidos não apresentarem optimizações. Assim, foi avaliado o impacto do suporte adaptativo através da duração da reconfiguração para as diferentes transições. Os testes efectuados permitem determinar qual o impacto da reconfiguração e como varia de acordo com o número de nós. As secções seguintes descrevem o ambiente de testes e os resultados obtidos.

6.7.1 Ambiente de Testes

A avaliação do caso de estudo consistiu na execução de testes à duração das transições descritas nas Secção 6.5. Cada transição é aplicada por uma estratégia e a utilização, ou não, da técnica de alteração dinâmica, aspecto que as distingue. Deste ponto de vista, as transições podem ser agrupadas. Assim, a Tabela 6.8 ilustra a divisão das cinco transições existentes em três grupos:

Transição	A	B	C	D	E
SetValue			x		
Add/RemoveProtocol	x			x	
ChangeProtocol/QoS		x			x

Tabela 6.8: Grupos de estratégias

O grupo *SetValue* apenas engloba as transições realizadas por uma estratégia local mas sem utilizar a técnica de alteração dinâmica da composição. Neste caso, é apenas uma, a transição C que considera a acção de reconfiguração *SetValue*. O grupo *Add/RemoveProtocol* engloba as transições realizadas por estratégias locais, utilizando a técnica de alteração dinãmica. Este grupo engloba duas acções: *AddProtocol* e *Removeprotocol*. Por fim, o último grupo, refere-se às transições aplicadas por estratégias globais e que utilizam a técnica de alteração dinâmica. O último grupo engloba as restantes acções: *ChangeProtocol* e *ChangeQoS*.

Os testes efectuados para cada um dos grupos consistiram na variação do número de nós participantes, desde um a quatro. Cada nó utiliza dois canais, um de controlo, para o suporte do elemento de reconfiguração, e um de dados, que serve as necessidades da aplicação, nomeadamente o envio ou recepção de posições da bola. A Figura 6.4 descreve as configurações dos canais.

A realização dos testes implicou a utilização de várias máquinas, cujas especificações se encontram resumidas na Tabela 6.9, numa rede local a 100 Mbps. Os componentes referem-se ao gestor da adaptação e ao monitor de contexto, que executavam no mesmo nó mas distinto dos restantes. Cada um dos nós intervenientes nos testes executava numa máquina distinta.

	Componentes	Nó 1	Nó 2	Nó 3	Nó 4
Processador	Pentium4	Pentium4	Pentium4	Pentium4	Pentium4
Frequência	3,2 GHz	3,2 GHz	2,8 GHz	2,8 GHz	3,2 GHz
Memória	1 GB	1 GB	224 MB	224 MB	0,99 GB
S.Operativo	WinXP	WinXP	WinXP	WinXP	WinXP

Tabela 6.9: Especificações das máquinas de acordo com o nó/componente

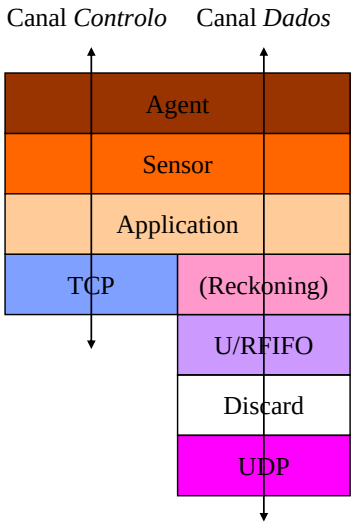

Figura 6.4: Duração das transições no agente de reconfiguração

6.7.1.1 Descrição dos Resultados

Os testes realizados mediram, para cada grupo e diferente número de nós, a duração do elemento de reconfiguração tanto no executor como no agente de cada nó. Foram efectuados três ensaios para cada teste, pelo que são apresentados valores médios.

Para cada um dos grupos foi efectuado um teste com um tamanho do sistema entre um nó e quatro nós. Todos os nós trocam mensagens relativas à posição da bola. Os nós estão identificados desde 1 a 4. Um teste com um nó, implica a participação do nó 1, com dois nós, dos nós 1 e 2, e assim sucessivamente. O nó 1 cumpria o papel de nó modificador do movimento da bola e, como tal, era o nó que gerava mensagens. As Tabelas 6.10 e 6.11 apresentam os valores médios, em segundos, para a duração da aplicação da reconfiguração para cada um dos grupos.

Os tempos apresentados resultaram da execução de três ensaios para cada teste e são o valor médio resultante dos valores de cada um dos nós intervenientes. Estes valores, organizados por componente, executor ou agente da reconfiguração, e pelo número de nós, são apresentados em gráficos. Os valores referem-se ao intervalo de tempo desde a primeira directiva recebida (nos agentes) ou enviada

	1 nó	2 nós	3 nós	4 nós
SetValue	0,0045	0,0047	0,0066	0,0053
Add/RemoveProtocol	0,0173	0,0233	0,0250	0,0227
ChangeProtocol/QoS	0,0361	0,0383	0,0413	0,1095

Tabela 6.10: Resultados dos testes no agente de reconfiguração (segundos)

	1 nó	2 nós	3 nós	4 nós
SetValue	0,0133	0,0107	0,0152	0,0112
Add/RemoveProtocol	0,0437	0,0509	0,0498	0,0394
ChangeProtocol/QoS	0,0417	0,0446	0,0523	0,1246

Tabela 6.11: Resultados dos testes no executor de reconfiguração (segundos)

(no executor) até à conclusão da estratégia. A Figura 6.5 apresenta o gráfico para o componente agente de reconfiguração.

No gráfico apresentado, o grupo *SetValue*, estratégia local e sem técnica de alteração dinâmica, apresenta a menor duração para a aplicação da reconfiguração. A duração mantém-se constante para sistemas com diferente número de nós, dado que a estratégia é local, não dependendo dos outros nós. A duração é a mais baixa de todas, visto que não é necessário criar um novo canal, para reconfigurar. De seguida, está o grupo *Add/RemoveProtocol*, que apresenta uma duração intermédia das três. Este grupo, apesar de também utilizar uma estratégia local, na aplicação da reconfiguração necessita de criar um novo canal, pelo que, comparativamente com o grupo anterior, a duração aumenta. A duração da aplicação da reconfiguração é estável, não variando com o aumento do número de nós do sistema. Por fim, o último grupo, *ChangeProtocol/QoS*, já considera uma estratégia global, em que existe coordenação dos nós, e apresenta a maior duração da aplicação da estratégia. O intervalo de tempo necessário para aplicar a reconfiguração aumenta com o número de nós, dado que os nós têm de esperar uns pelos outros.

A Figura 6.6 apresenta o gráfico para o componente executor da reconfiguração. No caso do grupo *SetValue*, a aplicação da reconfiguração apresenta a menor duração. Deve-se obviamente à ausência de coordenação e de não ser necessário criar um novo canal. A duração é estável com o aumento do número de nós do

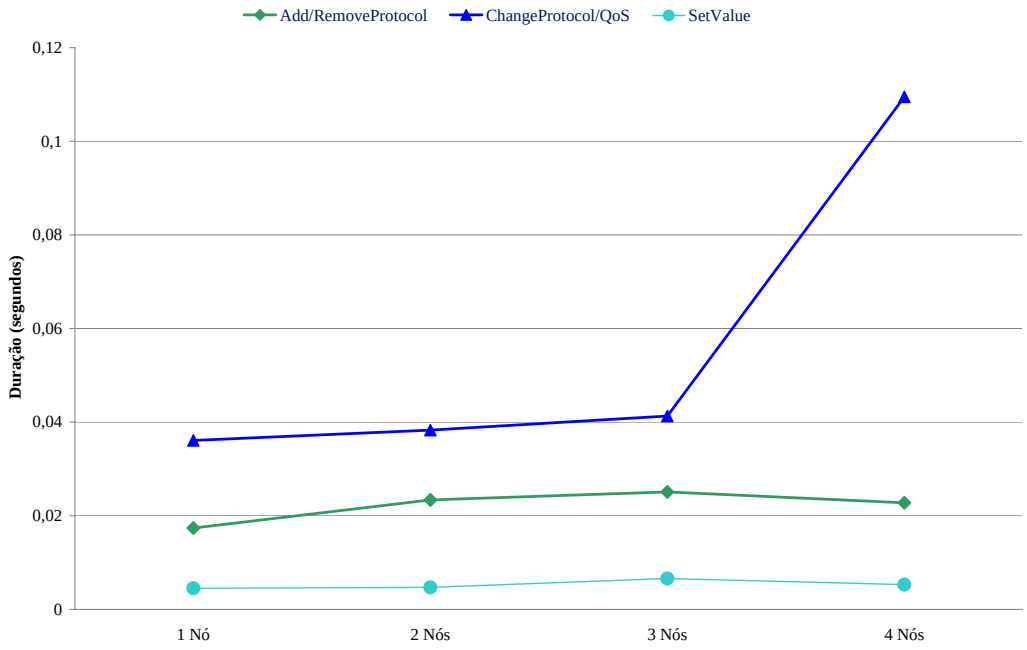

Figura 6.5: Duração das transições no agente de reconfiguração

sistema, à semelhança do agente de reconfiguração. O grupo *Add/RemoveProtocol* apresenta uma duração superior à anterior, como seria de esperar, devido à utilização da técnica de alteração dinâmica. A duração mantém-se estável com o aumento do número de nós do sistema, dado que se trata de uma estratégia local. No entanto, o grupo *ChangeProtocol/QoS*, com coordenação entre os nós, apresenta valores de duração muito semelhantes ao grupo anterior, para testes com um, dois e três nós. Este impacto mínimo justifica-se pelo facto do introduzido pela criação de um novo canal ser relativamente alto, comparativamente ao atraso devido à coordenação. Apenas no caso em que já existem quatro nós, é que a duração da aplicação da reconfiguração aumenta. Este impacto deve-se ao peso já significativo do atraso introduzido pela coordenação, quando o número de nós é superior a três.

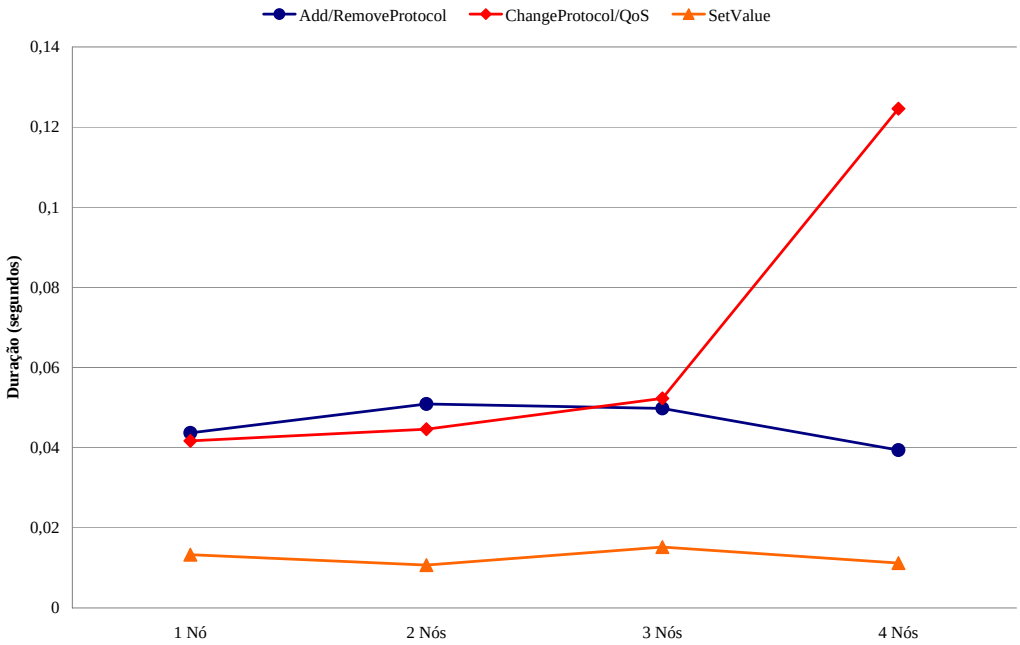

Figura 6.6: Duração das transições no executor da reconfiguração

Sumário

Neste capítulo foi descrito o caso de estudo construído para validar o protótipo apresentado no capítulo anterior. A política de adaptação e a definição do modelo construídas para o caso de estudo foram abordadas em pormenor, bem como os detalhes de concretização da aplicação de demonstração. Após a avaliação realizada neste capítulo, é possível concluir a dissertação no capítulo seguinte.

Capítulo 7

Conclusão e Trabalho Futuro

O desenvolvimento de aplicações adaptativas cientes do contexto permite lidar com as alterações do contexto de execução, aumentando o tempo de vida das aplicações e optimizando o seu desempenho. As metodologias orientadas a políticas facilitam o desenvolvimento e manutenção dessas aplicações, bem como permitem a reutilização do suporte adaptativo noutros contextos.

Nesta dissertação, foi proposta uma linguagem para a definição de políticas orientadas à adaptação de pilhas de protocolos, pressupondo uma arquitectura genérica para o suporte à construção de sistemas adaptativos. Foi descrito um protótipo da arquitectura de suporte, orientado a sistemas de comunicação e ciente do contexto. Foi abordado o enriquecimento da plataforma *Appia* para permitir a separação de aspectos funcionais e adaptativos necessária para a construção de sistemas orientados a políticas. Deste enriquecimento resultou a plataforma extendida *RAppia*.

Para validar o protótipo construído foi desenvolvido um caso de estudo, sendo apresentada a política de adaptação e o modelo de contexto correspondentes. O caso de estudo permitiu analisar o grau de dificuldade na definição de políticas e validar a arquitectura proposta. Desta avaliação, concluiu-se que a linguagem desenvolvida permite especificar políticas de adaptação de forma integral e adequada. Esta conclusão permitiu ainda consolidar a observação de que a linguagem é suficientemente genérica para ser utilizada noutros contextos. Os modelos de contexto e alvos da adaptação desenvolvidos estruturam de forma completa o suporte de contexto e reconfiguração para a especificação da política. Por fim,

a especificação de alto nível das políticas de adaptação, oferece uma separação clara entre a lógica da adaptação e a aplicação da mesma, através de diferentes estratégias. Esta separação permite ainda a reutilização da política em situações distintas.

Da construção do protótipo e caso de estudo resultaram várias conclusões relativas às limitações da plataforma *Appia* no âmbito adaptativo. Apesar de identificadas e colmatadas as limitações relacionadas com a ausência de tipificação e hierarquia a nível dos componentes da plataforma, existem outros obstáculos ao suporte da adaptação. Esses obstáculos referem-se à dificuldade de aplicar reconfigurações que alterem parcialmente a composição da pilha.

Como trabalho futuro, pretende-se melhorar os mecanismos de reconfiguração parcial da pilha, de forma a suprimir a técnica de alteração dinâmica nesses casos. Outro objectivo futuro é a extensão do modelo de contexto utilizado para especificar a informação observável, derivada e os eventos. Este modelo será também enriquecido com mecanismos para ligar as informações de contexto ao eventos. Estes mecanismos permitirão oferecer interrogações mais ricas e poderosas para o acesso à informação capturada. Pretende-se ainda desenvolver outras estratégias para aplicar as acções de reconfiguração, aumentando a variedade de estratégias possíveis de serem reutilizadas noutros contextos.

Bibliografia

Acharya, A., Ranganathan, M. & Saltz, J.H. (1997). Sumatra: A language for resource-aware mobile programs. In *MOS '96: Selected Presentations and Invited Papers Second International Workshop on Mobile Object Systems - Towards the Programmable Internet*, 111–130, Springer-Verlag, London, UK. 6

Bhatti, N.T., Hiltunen, M.A., Schlichting, R.D. & Chiu, W. (1998). Coyote: a system for constructing fine-grain configurable communication services. *ACM Trans. Comput. Syst.*, **16**, 321–366. 23

Carzaniga, A., Picco, G.P. & Vigna, G. (1997). Designing distributed applications with mobile code paradigms. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, 22–32, ACM Press, New York, NY, USA. 12

Chen, G. & Kotz, D. (2000). A survey of context-aware mobile computing research. Tech. rep., Hanover, NH, USA. 5

Coutaz, J., Crowley, J.L., Dobson, S. & Garlan, D. (2005). Context is key. *Commun. ACM*, **48**, 49–53. 8

Damianou, N., Dulay, N., Lupu, E. & Sloman, M. (2001). The ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, 18–38, Springer-Verlag, London, UK. 15

Eugster, P.T., Felber, P.A., Guerraoui, R. & Kermarrec, A.M. (2003). The many faces of publish/subscribe. *ACM Comput. Surv.*, **35**, 114–131. 7

Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B. & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, **37**, 46–54. 10

Grace, P., Coulson, G., Blair, G., Porter, B. & Hughes, D. (2006). Dynamic reconfiguration in sensor middleware. In *MIDSENS'06: Proceedings of the First International Worskshop on Middleware for Sensor Networks*, ACM, Melbourne, Australia. 62

Gupta, S.K.S. & Srimani, P.K. (1999). An adaptive protocol for reliable multicast in mobile multi-hop radio networks. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, 111, IEEE Computer Society, Washington, DC, USA. 20

Heinzelman, W.R., Kulik, J. & Balakrishnan, H. (1999). Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 174–185, ACM Press, New York, NY, USA. 20

Hiltunen, M.A., Schlichting, R.D., Ugarte, C.A. & Wong, G.T. (2000). Survivability through customization and adaptability: The cactus approach. vol. 1, IEEE. 23

Kawadia, V. & Kumar, P.R. (2003). A cautionary perspective on cross layer design. 22

Keeney, J. & Cahill, V. (2003). Chisel: A policy-driven, context-aware, dynamic adaptation framework. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, IEEE Computer Society, Washington, DC, USA. 17

Kwon, Y., Fang, Y. & Latchman, H. (2004). Performance analysis for a new medium access control protocol in wireless lans. *Wirel. Netw.*, **10**, 519–529. 22

Lopes, A. & Fiadeiro, L. (2005). Context-awareness in software architectures. In *Proocedings of the 2nd European Workshop in Software Architecture, EWSAM*, Springer-Verlag, Pisa, Italy. 35

Magee, J., Kramer, J. & Sloman, M. (1989). Constructing distributed systems in conic. *IEEE TSE 15*. 10

McCarthy, D.R. & Dayal, U. (1989). The architecture of an active database management system. In *ACM-SIGMOD International Conference on Management of Data*, ACM Press. 10, 43

McKinley, P.K., Sadjadi, S.M., Kasten, E.P. & Cheng, B.H.C. (2004). A taxonomy of compositional adaptation. Tech. Rep. MSU-CSE-04-17, Department of Computer Science, Michigan State University, East Lansing, Michigan. 10

Miranda, H., Pinto, A. & Rodrigues, L. (2001). Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of The 21st International Conference on Distributed Computing Systems (ICDCS-21)*, 707–710, IEEE Computer Society, Phoenix, Arizona, USA. 3, 23

Mocito, J., Rosa, L., Almeida, N., Miranda, H., Rodrigues, L. & Lopes, A. (2005). Context adaptation of the communication stack. In *ICDCSW '05: Proceedings of the Third International Workshop on Mobile Distributed Computing (MDC) (ICDCSW'05)*, 652–655, IEEE Computer Society, Washington, DC, USA. 51

Montanari, R., Tonti, G. & Stefanelli, C. (2003). Policy-based separation of concerns for dynamic code mobility management. In *COMPSAC '03: Proceedings of the 27th Annual International Conference on Computer Software and Applications*, 82, IEEE Computer Society, Washington, DC, USA. 12, 18

Montanari, R., Lupu, E. & Stefanelli, C. (2004). Policy-based dynamic reconfiguration of mobile-code applications. *Computer*, **37**, 73–80. 9, 12

Park, V.D. & Corson, M.S. (1997). A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, 1405, IEEE Computer Society, Washington, DC, USA. 20

Picco, G.P. (2000). Understanding code mobility. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, 834, ACM Press, New York, NY, USA. 12

Postel, J. (1981). Transmission control protocol. RFC 768. 21

Rigole, P., Berbers, Y. & Holvoet, T. (2004). Mobile adaptive tasks guided by resource contracts. In *MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, 117–120, ACM Press, New York, NY, USA. 10

Rodrigues, J., Miranda, H., Ventura, J. & Rodrigues, L. (2001). The design of rtappia. In *Proceedings of the Sixth IEEE International Workshop on Object-oriented Real-Time Dependable Systems*, IEEE, Rome, Italy. 50

Rodrigues, L., Miranda, H., Almeida, R., Martins, J., & Vicente, P. (2002). Strong replication in the globdata middleware. In *Proceedings of the Workshop on Dependable Middleware-Based Systems*, G96–G104, IEEE, Washington D.C., USA, (Suplemental Volume of the 2002 Dependable Systems and Networks Conference, DSN 2002). 50

Sloman, M. (1994). Policy driven management for distributed systems. *Journal of Network and Systems Management*, **2**, 333–360. 11

Sudame, P. & Badrinath, B.R. (2001). On providing support for protocol adaptation in mobile wireless networks. *Mob. Netw. Appl.*, **6**, 43–55. 9

Teixeira, S., Vicente, P., Pinto, A., Miranda, H., Rodrigues, L. & Martins, A., J. Rito-Silva (2002). Configuring the communication middleware to support multi-user object-oriented environments. In *Proceedings of*

the International Symposium on Distributed Objects and Applications (DOA), 965–980, Irvine (CA), USA. 50

Xylomenos, G. & Polyzoz, G. (1999). Tcp and udp performance over a wireless lan. In *Proceedings of the IEEE INFOCOM 1999*, 439–446, IEEE. 20, 22