Concretização e Avaliação de uma Plataforma de Suporte à Composição e Execução de Protocolos

(versão revista)

Alexandre Jorge Matos Pinto

Dissertação submetida para obtenção do grau de

MESTRE EM INFORMÁTICA

Orientador:

Luís Eduardo Teixeira Rodrigues

Júri:

Pedro Manuel Barbosa Veiga (Universidade de Lisboa)

José Orlando Roque Nascimento Pereira (Universidade do Minho)

André Osório e Cruz de Azerêdo Falcão (Universidade de Lisboa)

Luís Eduardo Teixeira Rodrigues (Universidade de Lisboa)

Setembro de 2004

Concretização e Avaliação de uma Plataforma de Suporte à Composição e Execução de Protocolos

(versão revista)

Alexandre Jorge Matos Pinto

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:

Pedro Manuel Barbosa Veiga (Universidade de Lisboa)

José Orlando Roque Nascimento Pereira (Universidade do Minho)

André Osório e Cruz de Azerêdo Falcão (Universidade de Lisboa)

Luís Eduardo Teixeira Rodrigues (Universidade de Lisboa)

Setembro de 2004

Resumo

O desenvolvimento de protocolos de comunicação pode ser simplificado através do uso de plataformas de suporte à composição e execução adequadas. Esta dissertação descreve a concretização de uma plataforma de suporte à composição e execução de protocolos concreta, o Appia, e faz a avaliação das suas diversas facetas. Nomeadamente, avalia a expressividade e eficácia dos mecanismos de suporte à composição assim como o desempenho do ambiente de execução.

Para suportar a avaliação, foi desenvolvida uma concretização de um serviço de comunicação em grupo com requisitos de composição complexos, os Grupos Ligeiros. A concretização deste serviço na plataforma Appia exigiu o desenvolvimento prévio de um sistema completo de comunicação em grupo oferecendo sincronia na vista, o qual é também descrito na dissertação. Para facilitar uma análise comparativa, e de modo a extrair ensinamentos que foram aplicados no desenvolvimento do Appia, o serviço de Grupos Ligeiro foi também concretizado sobre uma outra plataforma de suporte à composição e execução, em particular, sobre a plataforma Ensemble. A dissertação apresenta uma análise dos dois protótipos resultantes, extraindo indicações para o desenvolvimento de plataformas futuras.

PALAVRAS-CHAVE: Sistema distribuídos, Composição de protocolos, Ambientes de execução de protocolos

Abstract

Protocol composition and execution platforms can simplify the development of communication protocols. This thesis describes the development of a specific protocol composition and execution platform, the Appia, and performs an evaluation of its many characteristics. In particular it evaluates the composition support mechanisms in terms of their capabilities and ease of use. An evaluation of the execution environment performance is also made.

To support the evaluation, a group communication service with complex composition requirements was implemented. The service chosen was the Light-Weight Groups. To be able to implement this service on Appia, it was necessary to develop a group communication support with virtual synchrony. This is also described in the thesis. The Light-Weight Groups service was also implemented in another protocol composition and execution platform, the Ensemble, which allowed a comparative analysis between the two systems. The lessons learned during the service implementation in Ensemble, were used in the development of Appia. The thesis also presents an analysis of the two implemented prototypes, offering directions to the development of future platforms.

KEY-WORDS: Distributed systems, Protocol composition, Protocol execution environments

Agradecimentos

Este trabalho não teria sido concluído sem colaboração e ajuda de um conjunto de pessoas, às quais eu quero deixar aqui os meus agradecimentos. Antes de prosseguir, quero deixar aqui bem claro que o atraso na conclusão deste trabalho é da minha única e exclusiva responsabilidade.

A nível pessoal eu queria agradecer em primeiro lugar à minha mulher, pelo seu carinho, paciência, dedicação e incentivo, sem os quais este trabalho não teria sido concluído. A minha mãe foi uma das principais forças motrizes por detrás da escolha do caminho profissional que tomei e o seu constante incentivo foi muito importante para que este trabalho fosse concluído. Finalmente, queria deixar uma palavra de grande amor e carinho para o meu avô Manuel, que a conclusão deste trabalho possa servir como um pequeno tónico na luta que trava.

O Departamento de Informática da Faculdade de Ciências, o LaSIGE[1] em geral e os grupos Navigators e DIALNP[2] em particular, ofereceram o suporte necessário a que este trabalho fosse desenvolvido e os seus membros contribuíram para que este trabalho tenha sido finalmente concluído. Destes eu gostava de destacar em primeiro lugar o Professor Luís Rodrigues que sempre acreditou nas minha potencialidades e que fez tudo o que estava ao seu alcance para que este trabalho fosse concluído, apesar das adversidades e resistência encontradas. Gostava igualmente de destacar o Filipe Araújo que sempre me incentivou e ajudou e o Hugo Miranda cujo excelente trabalho de concepção do Appia é a base deste trabalho.

Este trabalho foi parcialmente financiado pelos projectos TOPCOM (FCT -

[1] Laboratório de Sistemas Informáticos de Grande Escala
[2] Distributed ALgorithms and Network Protocols

PRAXIS/P/EEI/12202/1998) e StrongRep (FCT - POSI/CHS/41285/2001)

Lisboa, Setembro de 2004

Alexandre Pinto

à João, à Luzia e ao Manuel

Índice

Lista de Figuras

Lista de Tabelas

1

Introdução

O desenvolvimento de aplicações distribuídas é cada vez mais exigente em relação aos serviços prestados pelo sub-sistema de comunicação. Por esta razão, a complexidade dos protocolos de comunicação tem também vindo a aumentar. Por exemplo, uma aplicação cooperativa multi-utilizador pode utilizar múltiplos canais de comunicação, com características distintas (Teixeira *et al.* , 2002), incluíndo serviços como comunicação em grupo fiável, ordenação de mensagens em difusão, serviços de filiação em grupo, ou protocolos de segurança.

Com a finalidade de simplicar a tarefa de desenvolver e configurar sistemas de comunicação complexos têm vindo a ser desenvolvidas plataformas de suporte à composição e execução de protocolos de comunicação. Existem diversas destas plataformas disponíveis, das quais pode-se destacar o Ensemble (Hayden, 1998) e Cactus (Cactus, 2004). Tipicamente, estas plataformas oferecem os seguintes mecanismos e serviços:

- Mecanismos que facilitam a decomposição de protocolos complexos de forma modular.
- Mecanismos que facilitam a configuração do sistema de comunicação, com base em componentes de software pré-existentes.
- Bibliotecas de utilitários que facilitam a codificação de novos protocolos, como a gestão de temporizadores ou mecanismos para facilitar a adição e remoção de cabeçalhos a mensagens.
- Um ambiente de suporte optimizado para a execução de composições de protocolos.

Esta dissertação descreve a concretização e avaliação da plataforma de suporte à composição e execução de protocolos *Appia*, tendo como ponto-de-partida uma especificação da mesma produzida por terceiros (Miranda, 2001). A plataforma Appia foi concebida com a intenção de facilitar o desenvolvimentos de sistemas de comunicação onde a actividade de vários canais de comunicação necessita de ser coordenada (por exemplo, para sincronizar fluxos de audio e video em aplicações multimédia). Para esse efeito introduz a noção de sessão partilhada e a funcionalidade de, em tempo de configuração, ser possível indicar, para cada componente do sistema de comunicação, quais as sessões a partilhar e o âmbito desta partilha.

1.1 Objectivos

O trabalho aqui descrito possui dois objectivos, que se complementam:

- Através da concretização de casos de estudo, aferir a adequabilidade dos mecanismos de composição propostos pelo Appia no desenvolvimento de serviços de comunicação com requisitos de coordenação multi-canal.
- Aferir a exequibilidade de concretizar estes mecanismos de forma eficiente, sem penalizar o desempenho da execução dos protocolos.

Estes objectivos são comuns, em parte ou na totalidade, a outros trabalhos efectuados, nomeadamente os descritos em (Sergio Mena *et al.* , 2003) e (Baldoni *et al.* , 2003).

1.2 Resultados

Foi realizada uma concretização da plataforma Appia usando a linguagem JAVA. Para suportar a avaliação da plataforma, foi desenvolvida uma concretização de um serviço de Comunicação em Grupo com requisitos de composição complexos, os *Grupos Ligeiros* (Guo & Rodrigues, 1997). A concretização deste serviço na plataforma

Appia exigiu o desenvolvimento prévio de um sistema completo de Comunicação em Grupo oferecendo *sincronia na vista*, o qual é também descrito na dissertação. Para facilitar uma análise comparativa, e de modo a extrair ensinamentos que foram aplicados no desenvolvimento do Appia, o serviço de Grupos Ligeiros foi também concretizado sobre uma outra plataforma de suporte à composição e execução, em particular, sobre a plataforma Ensemble (Pinto *et al.* , 2001). A dissertação apresenta uma análise dos dois protótipos resultantes, extraindo indicações para o desenvolvimento de plataformas futuras.

1.3 Estrutura da dissertação

A dissertação é composta por seis capítulos, com o seguinte conteúdo.

O Capítulo 2 apresenta uma panorâmica sobre algumas plataformas de suporte à composição e execução de protocolos que influenciaram o desenvolvimento do Appia. Para motivar os mecanismos de composição oferecidos por estas plataformas, o Capítulo é iniciado pelo levantamento de requisitos de composição de dois serviços de comunicação concretos.

O Capítulo 3 descreve o sistema Appia e a sua concretização. São também apresentados e descritos os protocolos de suporte à Comunicação em Grupo fiável desenvolvidos para o sistema Appia.

O Capítulo 4 apresenta as três concretizações do caso de estudo, o serviço de Grupos Ligeiros, sobre três plataformas diferentes, nomeadamente sobre o Horus (van Renesse *et al.* , 1996), Ensemble e Appia. As duas últimas destas concretizações foram desenvolvidas no âmbito desta dissertação.

O Capítulo 5 apresenta uma avaliação das concretizações efectuadas. Começa por dar uma avaliação qualitativa, em que são expostas de um ponto de vista conceptual as principais vantagens/desvantagens das plataformas utilizadas. De seguida é apresentada uma avaliação quantitativa, baseada em testes efectuados usando os protótipos desenvolvidos. Por último são referidas e comentadas avaliações aos protótipos feitas

por terceiros.

O Capítulo 6 conclui este documento e propõe trabalho futuro.

2

Composição de protocolos

O capítulo começa por definir o que se entende por protocolos com requisitos complexos em termos de composição, e serão dados dois exemplos, como forma de concretizar o problema. Depois será apresentado um resumo das várias plataformas de suporte à composição existentes, dando particular ênfase à que será usada para concretizar um dos protocolos com requisitos complexos.

2.1 Requisitos complexos em termos de composição

Certos sistemas, para a sua correcta concretização, impõem um conjunto de requisitos complexos à plataforma de suporte à comunicação usada para os concretizar. Vamos concentrar a nossa atenção nas plataformas que oferecem um conjunto de protocolos, as vezes referidos como micro-protocolos, em módulos separados que podem ser compostos de forma a oferecer propriedades mais complexas. Estes sistemas têm substituído os anteriores sistemas monolíticos que ofereciam todas as propriedades em um único bloco. Embora estes fossem construídos por vários módulos separados para diversas funcionalidades, tinham inúmeras interdependências e estavam de tal forma interrelacionados que a modificação da sua composição ou a sua reutilização era praticamente impossível. A predominância actual das plataformas com suporte à composição de protocolos deve-se: à circunscrição dos problemas ("dividir para conquistar"); à maior flexibilidade de configuração e adaptação; e à maior capacidade de adicionar e modificar os protocolos e correspondentes propriedades. Na realidade muitas das plataformas actuais com capacidade de composição de protocolos são descendentes directos de versões monolíticas.

Vulgarmente, os requisitos impostos sobre as plataformas recaem sobre: as propriedades oferecidas; a sua capacidade de adaptação e configuração; a facilidade de desenvolvimento, entre outros. Mas certos protocolos impõem requisitos específicos às capacidades de composição de protocolos, ou seja, a forma como estes são organizados e interagem entre si. Considere-se que se tratam de requisitos mais complexos do que o vulgar. A forma mais vulgar de composição é a pilha, em que os diversos protocolos são colocados uns sobre os outros formando uma pilha, e em que um protocolo para oferecer as suas propriedades ao protocolo que se encontra por cima utiliza as propriedades oferecidas pelos protocolos que se encontram por baixo. Esta composição não serve para todos os casos. Por exemplo, a capacidade de partilhar uma instância de um protocolo entre várias pilhas pode ser muito útil para permitir a sincronização entre essas várias pilhas, ou garantir uma propriedade transversal às pilhas. Essa sincronização pode ser assim alcançada de forma muito mais elegante do que se fosse concretizada na aplicação, o que às vezes nem sequer é possível.

Ao analisar-se alguns destes protocolos com requisitos complexos e a sua concretização em plataformas de suporte à composição de protocolos específicas, poderemos retirar ilações sobre as capacidades das plataformas em si e as necessidades que devem ser abordadas no desenvolvimento de novas plataformas. Embora se foque mais atenção nas características de composição de protocolos, as plataformas também serão avaliadas noutras vertentes, podendo ser retiradas conclusões sobre a facilidade de utilização ou de criação de novos protocolos.

2.2 Exemplos

Como exemplos de sistemas com requisitos complexos no que diz respeito à capacidade de composição dos protocolos da plataforma utilizada, serão apresentados primeiro os Grupos Ligeiros (Guo & Rodrigues, 1997) (Rodrigues & Guo, 2000) e de seguida o *MOOSCo* (Teixeira *et al.* , 2002).

2.2.1 Grupos Ligeiros

O objectivo dos Grupos Ligeiros é oferecer uma melhoria do desempenho de um serviço de Comunicação em Grupo em ambientes com vários grupos de filiação semelhante, ou seja, cuja maioria dos membros pertence aos mesmos grupos. Embora a melhoria deva ser significativa nestes ambientes, isto deve ser alcançado sem uma degradação significativa do desempenho do sistema em outros ambientes.

A optimização é alcançada através da partilha de recursos entre os vários grupos com filiações semelhantes. No entanto, a partilha de recursos pode originar um incremento da interferência entre os grupos que partilham os mesmos recursos, que por sua vez pode dar origem a uma degradação do desempenho. Para obter os melhores resultados tem que se chegar a um compromisso, em que se maximiza a partilha, procurando ao mesmo tempo limitar a interferência.

O conceito Grupos Ligeiros original (Guo & Rodrigues, 1997) foi desenvolvido assumindo-se que não ocorreriam partições na rede. Uma partição na rede é um estado desta, causado por uma falha de comunicação, que origina dois ou mais subconjuntos de nós que conseguem comunicar com outros nós no mesmo subconjunto, mas que perderam toda a comunicação com os nós dos outros subconjuntos. Mais tarde o conceito de Grupos Ligeiros foi extendido de forma a suportar estas ocorrências (Rodrigues & Guo, 2000). A descrição seguinte segue a mesma ordem, ou seja, descreve primeiro o conceito básico, e depois como este pode ser extendido de forma suportar as partições de rede.

No fim, serão descritas as necessidades em termos de composição que o sistema impõe. É neste aspecto que o serviço se destaca de outros e o torna adequado a este trabalho.

2.2.1.1 Conceito Básico

Um dos conceitos principais de um grande número de serviços de Comunicação em Grupo é o modelo de *sincronia na vista* (Birman & van Renesse, 1994) (Schiper & Ricciardi, 1993). A *sincronia na vista* é um modelo que disponibiliza, a todos os mem-

bros do grupo, informação coerente sobre os membros que constituem o grupo. Esta informação é chamada de *vista*. O modelo garante que todos os membros activos recebem todas as vistas e que estas são entregues na mesma ordem em todos eles. É igualmente garantido que qualquer mensagem enviada no contexto de uma determinada vista é entregue, pelo menos, a todos os membros da vista actual que farão parte da vista seguinte. Em alguns sistemas existe a garantia que a mensagem é entregue antes da vista seguinte. Deste modo, todas as mensagens podem ser associadas com a vista na qual foram enviadas porque existe a garantia que serão entregues antes de próxima vista. Para concretizar esta garantia o modelo necessita de executar um protocolo de escoamento que garanta que nenhum processo envie novas mensagens durante o processo de mudança de vista. Se tal acontecesse, seria impossível garantir que todos os membros já tinham entregue todas as mensagens da vista.

Os Grupos Ligeiros é um serviço transparente e dinâmico, cujo objectivo é a partilha de recursos, de forma a melhorar o desempenho em ambientes onde existam muitos grupos. Isto é alcançado tornando os grupos originais virtuais, ou *grupos ligeiros* (LWGs), e projectando-os num único grupo de suporte, ou *grupo pesado* (HWG). O serviço de Grupos Ligeiros deve preservar, na integra, o interface de grupos original. De salientar que a partilha de recursos também pode induzir perdas de desempenho. Estas resultam da interferência gerada entre os *grupos ligeiros* projectados no mesmo *grupo pesado*. Estas tendências opostas obrigam a que se alcance um compromisso entre partilha de recursos e redução da interferência de modo a alcançar o melhor desempenho possível.

O funcionamento do serviço de Grupos Ligeiros resume-se ao controlo e manutenção da colecção de *grupos pesados*, e ao estabelecimento e monitorização das projecções entre os *grupos ligeiros* e os *grupos pesados*. Quando um *grupo ligeiro* é criado o serviço de Grupos Ligeiros deve verificar se existe um *grupo pesado* disponível que contenha todos os membros do novo *grupo ligeiro*. Se existir tal *grupo pesado* é então estabelecida uma projecção entre eles, caso contrário é criado e adicionado à colecção de *grupos pesados* um novo. Como não existem restrições, nem é possível prever, as mudanças que podem ocorrer nas composições dos grupos, uma projecção que poderia ser adequada num determinado momento pode deixar de o ser, à medida que

o sistema vai evoluindo. Para manter a eficiência do sistema as projecções devem ser dinâmicas, ou seja, devem poder mudar ao longo de tempo de modo a adequarem-se à evolução do sistema. Estas mudanças de projecção só podem ser alcançadas utilizando um protocolo de mudança, que permita de forma transparente mudar um *grupo ligeiro* de um *grupo pesado* para outro.

Assim podem ser identificadas três tarefas principais de um serviço de Grupos Ligeiros:

- preservar o interface de *sincronia na vista* dos *grupos pesados* às aplicações dos *grupos ligeiros*;
- estabelecer, avaliar e decidir sobre a adequação das projecções existentes entre *grupos ligeiros* e *grupos pesados*, de forma a satisfazer os requisitos definidos;
- executar, quando necessário um *protocolo de mudança* que permita mudar de forma transparente a projecção de um determinado *grupo ligeiro*.

O estabelecimento e monitorização das projecções é executado segundo três regras básicas:

Partilha Como o objectivo máximo do serviço é melhorar o desempenho através da partilha de recursos, *grupos ligeiros* com composições semelhantes devem ser projectados no mesmo *grupo pesado*;

Interferência Para minimizar a interferência os *grupos ligeiros* devem ser projectados num *grupo pesado* com uma composição semelhante;

Redução Devido à evolução do sistema, é possível o aparecimento de *grupos pesados* sem nenhum *grupo ligeiro* projectado, que devem ser removidos do sistema.

A avaliação destas regras é feita usando unicamente dados e uma heurística locais.

A concretização de um serviço de Grupos Ligeiros, necessita que as projecções entre *grupos ligeiros* e *grupos pesados* sejam guardadas de forma a poderem ser acedidas

por todos os intervenientes no sistema. Isto é alcançado por um *Serviço de Nomes* externo, que pode ser acedido para obter a projecção de um qualquer *grupo ligeiro*, ou para informar de uma mudança efectuada nessa projecção.

2.2.1.2 Grupos Ligeiros com suporte a Partições

Primeiro temos que definir uma *partição*. De forma simples pode-se definir como uma mudança na infra-estrutura de comunicação que origina uma topologia da rede na qual alguns dos nós, que eram previamente acessíveis deixam de o ser, relativamente a um subconjunto dos nós. Isto normalmente gera "ilhas", ou *partições*, de nós que comunicam entre si, mas não com o resto dos nós no sistema. Esta ocorrência pode dever-se a falhas intencionais, ou não, de nós ou ligações. Tal como podem acontecer, também podem ser resolvidas, levando a reunificação de partições. Quanto maior a rede maior a probabilidade destas partições ocorrerem.

Para usarmos o serviço de Grupos Ligeiros em redes de larga escala geográfica, a tolerância a partições revela-se de grande importância. No entanto esta capacidade de tolerância levanta problemas acrescidos ao serviço de Grupos Ligeiros. O primeiro problema é a detecção dos *grupos ligeiros* de outras partições após estas terem sido reunificadas. Por outras palavras, o serviço deve ter mecanismos que permitam detectar outros membros do *grupo ligeiro* que estavam activos numa outra partição e aos quais o acesso foi reestabelecido. Este problema é agravado pelo facto de que durante a partição as projecções podem ter sido modificadas, e como a coordenação era impossível, estas podem ser diferentes consoante a partição. Assim após a reunificação das partições, para se poder reunificar o *grupo ligeiro*, deve-se conciliar as projecções. Para isto, usa-se a seguinte estratégia (Rodrigues & Guo, 2000):

1. Depende-se de um *grupo pesado* com suporte a partições para fornecer *detecção de falhas*, recuperação de partições ao nível dos *grupos pesados*, e comunicação com *sincronia na vista*. Assim, evita-se ter que concretizar novamente ao nível dos *grupos ligeiros* estes protocolos. Estes mecanismos são transparentes para o serviço de Grupos Ligeiros permitindo o seu funcionamento em todos os ambientes suportados por esses protocolos sem necessidade de modificação dos Grupos Ligeiros.

2. Depende-se de um Serviço de Nomes externo para fornecer as projecções actualizadas. Este Serviço de Nomes tem que ser concretizado usando servidores distribuídos, de forma a poder continuar a funcionar perante partições da rede.

3. Usando estes dois mecanismos, após uma reunificação de partições ao nível dos *grupos pesados*, devem ser tomados os seguintes passos:

   1º Passo Vistas concorrentes de um *grupo ligeiro* projectadas em diferentes *grupos pesados* ficam conscientes da existência umas das outras.

   2º Passo Todos os *grupos ligeiros* que satisfazem o passo anterior devem conciliar as suas projecções, de forma a ficarem projectadas no mesmo *grupo pesado*.

   3º Passo Após todas as vistas concorrentes de um *grupo ligeiro* ficarem projectadas no mesmo *grupo pesado*, elas detectam-se umas as outras usando mecanismos locais ao *grupo pesado*.

   4º Passo Vistas concorrentes de um *grupo ligeiro* projectadas no mesmo *grupo pesado*, unem-se numa única vista.

Apesar do Serviço de Nomes se revestir de pouca importância ao assumir-se a inexistência de partições, ele ganha um relevo acrescido quando o suporte a estas se torna necessário. Isto porque o Serviço de Nomes é o local onde as diferentes projecções de um *grupo ligeiro* podem ser detectadas. Para ele próprio suportar partições, o Serviço de Nomes tem que ser concretizado como um grupo de servidores distribuídos. Deste modo, quando as partições são reunificadas, o Serviço de Nomes tem que ter um mecanismo que permita conciliar as projecções que as suas várias réplicas têm guardado, detectando vistas concorrentes. Quando estas vistas concorrentes são detectadas, o Serviço de Nomes deve notificar os correspondentes *grupos ligeiros*.

2.2.1.3 Necessidades de composição

O serviço de Grupos Ligeiros deve ser concretizado como uma nova camada, ou protocolo, que seria colocada nas várias composições de protocolos correspondentes aos diferentes grupos. Esta camada marcaria a fronteira entre os *grupos ligeiros* e os

grupos pesados, sendo que conceptualmente um *grupo ligeiro* seria a composição desde a aplicação até à camada do serviço de Grupos Ligeiros, enquanto um *grupo pesado* seria a composição entre a camada do serviço de Grupos Ligeiros e a ligação ao meio de comunicação. Deste modo o serviço poderia ser visto como um encaminhador entre *grupos ligeiros* e *grupos pesados*, em que a escolha feita no encaminhamento é semi-dinâmica, visto que depende das projecções estabelecidas e estas só são avaliadas, e se necessário, corrigidas periodicamente e por vista. Esta concretização goza de um alto nível de flexibilidade inerente, visto que os *grupos ligeiros* projectados num *grupo pesado* podem, cada um deles, oferecer um subconjunto de propriedades diferentes, consoante a composição de protocolos que se encontra entre o serviço de Grupos Ligeiros e a aplicação. Por outro lado, a composição correspondente ao *grupo pesado* constitui as propriedades, os recursos, partilhados entre todos os *grupos ligeiros* projectados. Assim, esta concretização exige que a plataforma de suporte permita que uma camada seja partilhada entre várias composições, ou seja, que permita composições complexas.

2.2.2 MOOSCo

No início da descrição será apresentado o conceito por detrás do sistema. Depois serão apresentadas as possibilidades de concretização e dessa forma demonstrado que a solução ideal requer que a plataforma suporte composições de protocolos complexas.

2.2.2.1 Conceito

O sistema desenvolvido no âmbito do projecto *MOOSCo* (Teixeira *et al.* , 2002) destina-se a suportar ambientes multi-utilizador orientados aos objectos (MOOs), que é uma das vertentes dos sistemas interactivos multi-utilizador distribuídos. Nestes, existem vários utilizadores, geograficamente separados, que interagem em tempo-real utilizado normalmente a metáfora de uma *sala virtual*. Nestas salas os utilizadores podem interagir entre si, com objectos existentes na sala e com a própria sala. Por exemplo, podem entrar e sair da sala ou deslocar um objecto de um utilizador para outro.

A informação sobre estas interacções tem que ser partilhada, distribuída, entre os diversos intervenientes respeitando certas regras de coerência. Cada objecto presente na sala é caracterizado por um conjunto de atributos específicos, e consoante o tipo desses atributos certos requisitos são impostos sobre a comunicação, por exemplo protocolos de difusão de dados têm necessidades diferentes dos protocolos de difusão de som. De realçar que algumas interacções só necessitam de comunicação entre os intervenientes directos, enquanto outras necessitam de ser difundidas a toda a sala porque implicam uma mudança do "aspecto" da mesma.

Existem duas alternativas para resolver estas questões. A primeira passa por definir um sistema que oferece um conjunto de características de comunicação que satisfaz todos os requisitos das diferentes interacções e atributos, mesmo que para alguns destes o desempenho oferecido seja inferior ao possível. Por exemplo, se a modificação de certos atributos requer ordem causal e outros ordem total, pode-se utilizar sempre a ordenação total. A outra alternativa passa por utilizar um sistema configurável que adapta as características da comunicação às necessidades das interacções e atributos, utilizando um conjunto de componentes que oferecem determinadas características e compondo-os por atributo de forma a satisfazer os requisitos deste. No entanto, para manter a coerência do ambiente, as diferentes interacções que se efectuam no ambiente devem-se sincronizar para que o ambiente vá aparecendo coerente em todos os intervenientes. Isto significa que, embora cada atributo utilize o seu próprio mecanismo de comunicação, tem que existir algo transversal a estes que permita efectuar a sua sincronização.

2.2.2.2 Necessidades de composição

Para demonstrar as necessidades de composição do sistema utilizemos o exemplo presente em (Teixeira *et al.* , 2002). Considere-se um jogo simples, um "multi-user-dungeon". Nestes cada jogador é um animal que se move num universo virtual à procura de comida. O utilizador é representado por um objecto, um *avatar*, com três atributos: aparência, posição no ambiente e sentido para que está virado. O avatar pode executar uma única acção, comer. Iremos considerar um cenário em que existem

dois utilizadores numa sala com uma ventoinha e um caixote com comida. Como a ventoinha é fixa só tem um atributo relevante, a sua velocidade, enquanto que o caixote com comida tem dois atributos importantes: posição e quantidade de comida que tem dentro. Cada utilizador tem que se aproximar do caixote com comida e comer a comida que se encontra lá. Ao comer os avatar modificam a sua aparência, "engordam". De realçar que se dois utilizadores tentarem retirar a ultima porção de comida, só um deles o deve conseguir, o que implica que estas acções devem respeitar uma ordem global. Além disso, a ordem pela qual os atributos são modificados deve respeitar uma relação de causalidade. Por exemplo, se a quantidade de comida no caixote diminui, deve ser modificada a aparência do(s) avatar, estes devem "engordar". Por outro lado modificações na velocidade da ventoinha são independentes das modificações efectuados nos atributos de outros objectos.

Neste sistema, em cada utilizador existe uma réplica de cada objecto, pelo que os utilizadores têm que ser informados de todas as modificações efectuadas nos atributos desses objectos. Utilizemos o termo *canal* para descrever uma composição dos protocolos de comunicação, que oferece um determinado conjunto de propriedades de comunicação, por exemplo em termos da ordem da entrega da informação.

Uma solução possível seria utilizar um canal independente por atributo, o que resultaria em que cada atributo utilizava um canal próprio que oferecia as propriedades estritamente necessárias para esse atributo. Por exemplo, a difusão das alterações à velocidade da ventoinha não utilizaria uma ordem em particular, enquanto a difusão das alterações da quantidade de comida no caixote utilizaria uma ordem total para garantir que todos os utilizadores vêm essas modificações de forma coerente. No entanto, esta solução não permite que seja garantida a ordem causal das modificações entre os vários atributos. Por exemplo, não era possível garantir a ordem entre a ingestão de comida do caixote e a modificação da aparência do avatar.

Para resolver este problema poderia-se utilizar uma solução com um único canal para todos os atributos. Este canal teria que oferecer propriedades para satisfazer os requisitos mais fortes de entre os vários atributos. Neste exemplo teria que se oferecer ordem total, embora só alguns deles necessitem efectivamente de uma garantia tão

forte. Como os protocolos de comunicação que oferecem ordem total têm geralmente um desempenho inferior aos protocolos que oferecem só ordem causal, ou nenhuma ordem, o desempenho do sistema não seria o melhor.

Assim, a solução passa por criar composições de protocolos para os canais dos atributos, em que alguns dos atributos partilham alguns dos protocolos de forma a garantirem a coerência necessária, como a causalidade das modificações de certos atributos. Um aspecto partilhado seria o protocolo que garante a difusão fiável. Como o atributo velocidade da ventoinha não tem mais nenhum requisito, o seu canal teria apenas este protocolo. Os restantes atributos, para garantirem a relação de causalidade entre eles, partilhariam um protocolo que garantisse ordem causal. Já a difusão da acção de comer, por parte dos utilizadores, utilizaria a ordem total. Como os movimentos dos avatar são sujeitos a regras, é possível extrapolar em parte as suas movimentações, permitindo assim não comunicar todos os movimentos, só os mais significativos, reduzindo a quantidade de informação a difundir. Isto é alcançado pelo protocolo "dead-reckoning". Com isto podem existir pequenas inconsistências na posição do avatar, e para garantir que quando um avatar come ele se encontra junto ao caixote da comida, existe um outro protocolo, "force-proximity". A composição descrita é apresentada na Figura 2.1

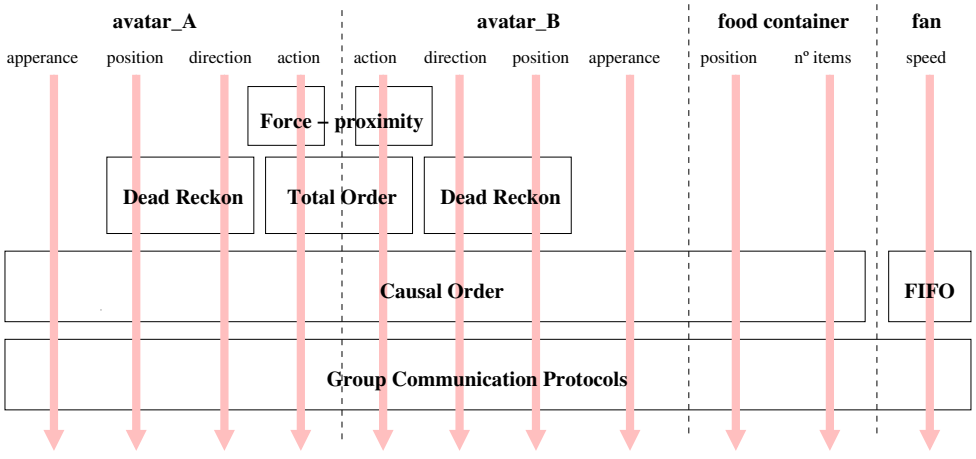

Figura 2.1: Composição com canais partilhados no sistema MOOSCo

2.3 Plataformas de suporte à composição de protocolos

Após uma breve apresentação de algumas das plataformas de suporte à composição de protocolos existentes, será apresentado em maior detalhe a primeira plataforma usada, o Ensemble. Devido ao seu desenvolvimento ter sido parte importante do trabalho efectuado, a apresentação da plataforma Appia será feita ainda em maior detalhe, pelo que ficará para o Capítulo 3.

2.3.1 x-Kernel

O *x-Kernel* (Hutchinson & Peterson, 1991) é um conjunto de utilitários e bibliotecas, oferecendo abstracções para o encapsulamento dos protocolos e mecanismos bastante optimizados para algumas das operações que os protocolos executam com mais frequência. A existência destas bibliotecas optimizadas para as operações mais frequentes é o principal motivo para os excelentes resultados de desempenho obtidos pelo x-Kernel. O sistema permite a composição de protocolos como um grafo, sendo que este é concretizado em tempo de compilação.

Existem três entidades básicas: protocolos, cuja principal responsabilidade se prende com o encaminhamento das mensagens ao longo do grafo; sessões, criadas em tempo de execução, contêm as estruturas de dados e código que fornecem as propriedades de uma instância de protocolo; mensagens, contêm a informação a comunicar e cabeçalhos, e são manipuladas quer por protocolos como por sessões.

As mensagens percorrem o grafo de maneira distinta consoante o percorrem no sentido ascendente ou descendente. Assim, no sentido descendente as mensagens vão a todas as sessões dos protocolos que se encontram no grafo nesse sentido, sendo que estas não necessitam de ter conhecimento da constituição do grafo, invocando uma função *push* genérica, que se encontra ligada à sessão que se encontra em baixo. Já no sentido ascendente, em cada protocolo as mensagens são entregues ao próprio protocolo que decide então a qual das suas sessões deve entregar seguidamente a mensagem e a qual protocolo a mensagem deve ser entregue após isso. Esta decisão baseia-se em informação que se encontra na mensagem e implica que os protocolos saibam quais

os protocolos que se encontram por cima, e que cada um destes tenha um identificar único conhecido.

De referir que existem também os chamados protocolos virtuais, que não têm sessões e servem unicamente para encaminhar as mensagens no grafo. Além da troca de mensagens a comunicação entre protocolos pode utilizar a invocação de funções de controlo sobre os protocolos imediatamente abaixo, que na sua execução podem invocar as funções de controlo do protocolo inferior.

No global, o x-Kernel oferece grande flexibilidade de configuração, pelos protocolos decidirem em tempo de execução o protocolo ao qual a mensagem vais ser entregue a seguir, e desempenho, pelas bibliotecas optimizadas. Isto é alcançado em detrimento da capacidade de reutilização, já que os protocolos têm que ter conhecimento sobre quais os protocolos que podem estar por cima deles. Além disso no sentido descendente não é possível mudar o encaminhamento das mensagens sem ter um profundo conhecimento da concretização do grafo. Já no sentido ascendente, se o encaminhamento depender dos dados colocados pela aplicação e não pelo protocolo corrente, por exemplo informação multimédia, o x-Kernel não oferece nenhum mecanismo para que a aplicação especifique esse encaminhamento desejado.

2.3.2 Coyote/Cactus

O *Coyote/Cactus* [1] (Bhatti *et al.* , 1998) é um sistema que suporta composições horizontais ou paralelas, em que as mensagens são processadas paralelamente pelos diversos protocolos (ou micro-protocolos), em clara oposição às composições verticais ou hierárquicas suportadas por quase todos os outros sistemas, em que as mensagens são processadas pelos protocolos de forma sequencial. Na realidade o sistema usa as duas formas de composição, sendo que a utilização de uma ou outra, depende do nível de granularidade. A Figura 2.2 (Cactus, 2004) apresenta a arquitectura do Cactus. Existem

[1] Não foi encontrado uma definição que esclareça cabalmente a diferença entre os dois, sendo ambos os nomes são usados regularmente na bibliografia existente. Após consultar (Cactus, 2004) fica-se com a impressão que Coyote é a plataforma de composição de protocolos do Cactus, pelo que seria esse o relevante no contexto deste trabalho, mas isso não é claro pelo que ambas as nomenclaturas podem ser usadas.

dois níveis de granularidade:

grossa, em que os protocolos, chamados de *protocolos compostos*, são agrupados de forma hierárquica (vertical) e comunicam entre si através de *mensagens*, que correspondem a uma colecção de atributos.

fina, em que os protocolos compostos são concretizados por vários *micro-protocolos* que operam de forma cooperativa e paralela, comunicando entre si através de eventos e memória partilhada.

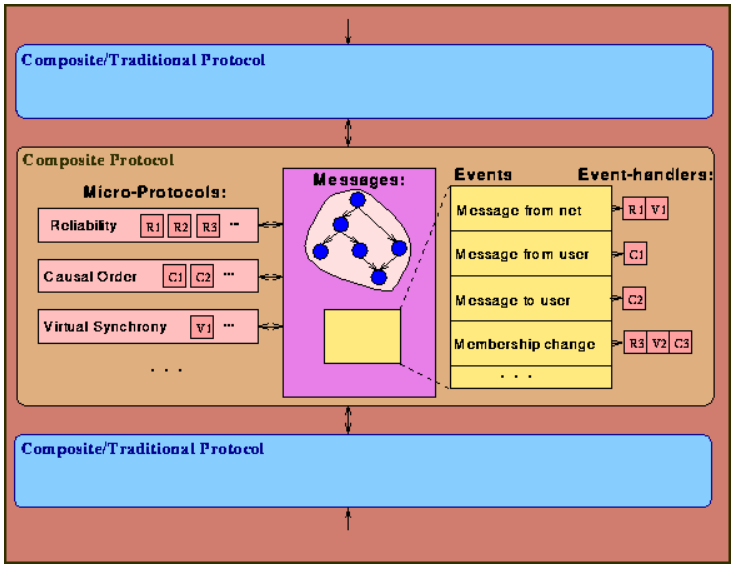

Figura 2.2: Arquitectura com dois níveis de granularidade do Cactus (Cactus, 2004)

Ao nível fino, em que o sistema é orientado aos eventos, cada micro-protocolo pode adicionar novos eventos em tempo de execução. Cada micro-protocolo regista quais os eventos que pretende receber e a função para os processar. As mensagens quando chegam ao sistema são colocadas numa zona partilhada por todos os micro-protocolos e estes são notificados da ocorrência por um evento. Uma mensagem só sai do protocolo composto quando todos os micro-protocolos tiverem manifestado a sua concordância, ou seja, quando a tiverem tratado. De notar que a plataforma não oferece mecanismos para o controlo da concorrência, ficando os programadores dos micro-protocolos com toda a responsabilidade nesse campo.

A composição paralela oferece uma grande flexibilidade, permitindo suportar interacções complexas entre os protocolos. Só que estes para oferecerem um resultado

coerente de forma simples, acabam por estabelecer fortes relações de dependência entre si, o que prejudica sua reutilização.

Ao nível grosso, os protocolos compostos são organizados de forma hierárquica seguindo o modelo do x-Kernel, do qual pode-se dizer que o Cactus descende. Os protocolos compostos em si têm como principal responsabilidade efectuar a comunicação com os protocolos compostos que lhe estão adjacentes, através da troca de mensagens. Quando uma mensagem é recebida de outro protocolo composto, ela tem que ser entregue aos micro-protocolos para estes a processarem. Quando estes terminarem o processamento, a mensagem tem que ser enviada para o protocolo composto seguinte. A determinação de qual é esse protocolo composto seguinte faz parte das atribuições de um protocolo composto. Tal como no x-Kernel, o protocolo tem que ter conhecimento da composição.

2.3.3 Horus

O *Horus* (van Renesse *et al.* , 1996) é uma plataforma de suporte à Comunicação em Grupo. A composição tendencial é a vertical, ou pilha. Esta é definida em tempo de execução e todas as camadas/protocolos têm que concretizar um conjunto de funções definidas, para manter um alto nível de modularidade. Em termos sintácticos as camadas podem ser organizadas livremente na pilha, independentemente dos resultados obtidos.

O sistema é totalmente orientado aos eventos, ou seja, toda a interacção é feita através de eventos que percorrem a pilha. No entanto, estes estão todos definidos, sendo impossível a adição de novos eventos, limitando desta forma a possibilidade de extender o sistema com novos protocolos. Esta limitação prende-se essencialmente com a maximização da capacidade de reutilização dos protocolos em detrimento da concretização de novos protocolos, uma posição radicalmente oposta à tomada por exemplo no Cactus. De realçar a existência do protocolo FAST, que perante um cenário temporário em que as propriedades oferecidas por um conjunto dos protocolos da pilha não necessitam de ser impostas, efectua um encaminhamento inteligente dos eventos pela pilha, que evita que os eventos passem por esses protocolos. Diminui-se assim

o tempo de processamento.

Embora a pilha seja a composição preferencial, o sistema suporta outras composições, com partilha de protocolos entre pilhas. Considerados excepções, existem alguns exemplos de utilização destas pilhas no Horus. Por exemplo, na resolução da dificuldade em escalar que a maior parte dos protocolos de Comunicação em Grupo padece. A solução passa pela hierarquização dos grupos de grande dimensão numa árvore, que é gerida pelo protocolo PARCLD. Este define duas "sub-pilhas" por baixo: uma dedicada à comunicação do nó com os seus pares e outra à comunicação com os nós de nível inferior. O encaminhamento dos eventos para as duas "sub-pilhas" é efectuado pelo protocolo, sendo o seu funcionamento transparente para as outras camadas, favorecendo a reutilização, mas obrigando a camada a interpretar todos os eventos que circulam na pilha.

2.3.4 Bast

O *Bast* (Garbinato *et al.* , 1996) destaca-se por utilizar a herança simples como mecanismo principal da composição dos protocolos. Um protocolo que depende de outro extende a classe que concretiza esse protocolo. Isto significa que a ligação entre os protocolos, a composição, é efectuada em tempo de compilação, efectuando-se a sua validação antecipadamente. No entanto, fica limitada a capacidade de em tempo de execução utilizar uma composição dinâmica, limitando as capacidades de configuração.

Uma característica possibilitada pelo uso da herança é a aplicação aceder directamente aos protocolos presentes na composição. Isto advém de cada protocolo extender os que utiliza, pelo que o protocolo de topo extende todos os outros, tendo por isso os métodos que estes ofereciam, que podem ser acedidos directamente.

O sistema utiliza um padrão de desenho, que à semelhança do x-Kernel define duas super-classes básicas: a PROTOOBJECT que define o protocolo e é responsável pela ligação em tempo de compilação aos protocolos de que depende; e a PROTOALGO que corresponde a concretização de um algoritmo. Tal como no x-Kernel, a PROTOOBJECT

pode em tempo de execução seleccionar entre as PROTOALGO existentes para o seu protocolo, oferecendo um grande nível de flexibilidade e adaptação dinâmica ao ambiente. O encaminhamento ao longo da composição é decidido por camada, o que acaba por se traduzir num consumo desnecessário de recursos.

2.3.5 Groupz

O *Groupz* (Pereira & Oliveira, 1997) é um sistema que utiliza as potencialidades da linguagem em que foi codificado, o JAVA, para concretizar uma "rede activa", ou seja, uma rede em que as próprias mensagens contém os mecanismos para operar sobre os protocolos na origem e no destino, com o intuito de oferecer as propriedades pretendidas. Assim em vez das propriedades da comunicação serem definidas pela composição dos protocolos, passam a ser definidas pelos mecanismos presentes nas própria mensagens. Isto permite que na mesma composição, na mesma pilha, sejam oferecidas propriedades diferentes por mensagem.

O funcionamento das camadas é por isso diferente, limitando-se a invocar métodos do objecto em que a mensagem é transportada. São estes que modificam a mensagem e o estado das camadas para reflectir, ou impor, as propriedades pretendidas. Como é a aplicação que cria estes objectos, o programador da aplicação tem que ter um maior conhecimento de como é que cada camada oferece as suas propriedades. Isto significa uma diminuição da possibilidade de utilização transparente do sistema por programadores não especializados, mas em contrapartida um aumento da flexibilidade e da capacidade de adaptação rápida e dinâmica à mudança das características do ambiente e das necessidades da aplicação.

Uma vantagem deste sistema é que as aplicações ganham uma forma de poderem expressar para o sistema as características semânticas das mensagens trocadas.

2.3.6 Ensemble

O *Ensemble* (Hayden, 1998) é um sistema desenvolvido na Universidade de Cornell, usando a linguagem *Objective Caml* [2].

O Ensemble oferece uma plataforma de Comunicação em Grupo fiável com *sincronia na vista*, sendo este o único mecanismo de comunicação disponibilizado. Foi desenvolvido para permitir a optimização de execução de pilhas de protocolos avançados, sendo a composição em pilha a única composição de protocolos suportada.

A decisão de suportar apenas uma composição estritamente vertical de protocolos prende-se com funcionalidades avançadas que o Ensemble oferece, nomeadamente a possibilidade de se efectuar a validação formal dos protocolos e de se optimizar o processamento das mensagens. Ambas não podem ser usadas em tempo de execução.

O Ensemble é orientado aos eventos, ou seja, todas as interacções entre camadas ou com o núcleo do sistema (ex: pedido de temporizador) é feita através de eventos. Uma camada é desenvolvida e concretizada para receber, processar e potencialmente gerar novos eventos, em reposta a um evento. Os tipos de eventos possíveis, cerca de quarenta, já estão definidos e é problemático a adição ou remoção destes, porque o próprio núcleo interpreta os eventos, e executa funcionalidade específica para cada evento.

Uma característica única do Ensemble é que a aplicação não reside no topo da pilha, como em sistemas semelhantes, ou no modelo OSI. Em vez disso a aplicação, mais concretamente a interface de aplicação do Ensemble, encontra-se paralelo à pilha, comunicando com uma camada que se encontra a meio desta (Figura 2.3). Isto foi feito para optimizar o desempenho das mensagens da aplicação, que desta forma não têm que percorrer toda a pilha de comunicação.

A interface de aplicação consiste num conjunto de funções que permitem registar funções de chamada ("call-back"). Estas serão usadas pelo Ensemble para entregar as mensagens recebidas e para notificar certas ocorrências dentro do grupo, como por

[2] Recentemente foi disponibilizada uma versão em C, mas o trabalho sobre a versão original concretizada em OCAML já se encontrava concluído

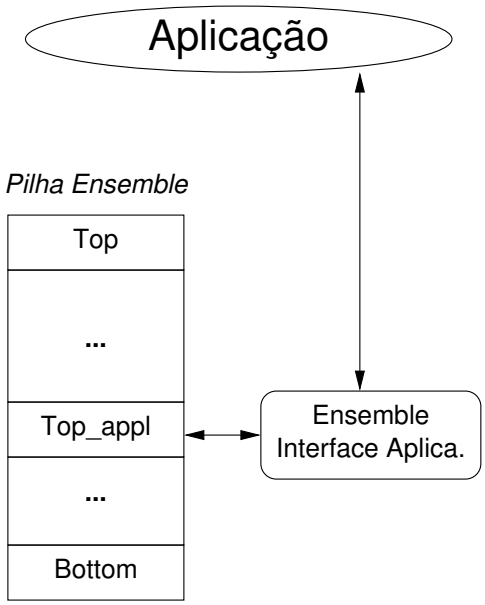

Figura 2.3: Colocação do interface de aplicação no Ensemble.

exemplo uma mudança de vista. Assim, as funções de chamada têm uma relação directa com certos eventos, embora estes sejam transparentes para a aplicação. Algumas destas funções podem retornar um conjunto de operações que a aplicação deseja efectuar, como por exemplo o envio de uma mensagem. Esta é a única forma da aplicação efectuar operações sobre o Ensemble. Para permitir efectuar uma operação sem ser em resposta directa a uma ocorrência particular, como a chegada de uma mensagem, uma das funções registadas ("heartbeat") é invocada periodicamente.

Outra característica importante do Ensemble é a linguagem na qual foi desenvolvido, o *Objective Caml* ou simplesmente OCAML. Trata-se de uma linguagem *funcional*, derivada da linguagem *ML*. Nesta linguagem é usada a Reciclagem Automática de Memória ("Garbage Collection"). Apesar desta simplificar a programação, vários cuidados devem ser tomados quando o desempenho é crucial. Nomeadamente, a má gestão que o reciclador automático de memória faz das variáveis de grande dimensão, obriga o programador a evitar o uso destas ou então a geri-las explicitamente. Isto aplica-se em particular às mensagens de dados trocadas entre os membros do grupo, cujo tamanho é potencialmente grande. Devido a isso, o Ensemble oferece ferramentas e mecanismos para gerir explicitamente uma estrutura de dados que encapsula estes grandes pedaços de dados. Além destas variáveis grandes, o uso de algumas

estruturas de dados, como as listas, também requer cuidados adicionais, pois tendem a gerar muitos elementos para o reciclador automático de memória processar. Assim, é aconselhável usar unicamente os mecanismos imperativos da linguagem, negando em parte o seu paradigma funcional. O Ensemble segue estas regras a alcança bons resultados, mas a utilização descuidada ou ingénua do OCAML e do Ensemble, leva invariavelmente a maus resultados em termos de desempenho.

Uma capacidade incomum do Ensemble é a de mudar a composição da pilha durante a execução. Tem igualmente vários serviços auxiliares e interfaces com varias outras linguagens como o C++ e o JAVA.

2.3.7 Comparação das capacidades de composição

Dos sistemas apresentados, três oferecem grande flexibilidade de composição. São eles o x-Kernel, o Cactus e o Bast. Todos permitem por exemplo que uma camada seja partilhada entre varias composições. No entanto, esta flexibilidade de composição é alcançada fazendo com que sejam as próprias camadas a decidir o caminho que a informação percorre ao longo da composição, pelo que elas têm que ser concretizadas com o suporte às diversas composições possíveis. Isto nem sempre é exequível, e quando o é gera fortes relações de interdependência que limita em grande parte a sua reutilização.

Os outros sistemas, o Horus, GroupZ e Ensemble, limitam as composições à vertical, ou seja, à pilha. O Ensemble permite apenas a composição estritamente vertical, não permitindo que uma camada seja partilhada entre várias pilhas. Já o Horus permite que uma camada seja partilhada entre várias pilhas, mas isto é considerado uma excepção e só pode existir uma única instância da camada partilhada. Nestes sistemas, a determinação do caminho percorrido pela informação dentro da composição fica a cargo da própria plataforma, permitindo um elevado grau de reutilização das camadas, que não necessitam de ser concretizadas tendo em consideração uma qualquer composição.

2.4 Sumário

Este capítulo apresenta dois exemplos de protocolos com requisitos complexos sobre a plataforma de composição de protocolos. Depois foi apresentado um resumo de algumas das plataformas de composição de protocolos existentes, sendo que uma delas, o Ensemble, é apresentada em maior detalhe porque será utilizada para efectuar a concretização de um dos protocolos complexos, os Grupos Ligeiros.

3
Appia

O *Appia* foi desenvolvido para tentar preencher lacunas existentes em outros sistemas, como o Ensemble, que permitissem suportar composições de protocolos mais complexas. A definição do sistema é feita em (Miranda, 2001), (Miranda & Rodrigues, 1999) e (Miranda *et al.* , 2001). Neste capitulo é apresentado um sumário da definição do sistema, seguido de uma descrição da concretização efectuada, centrada nos aspectos mais relevantes do sistema. Após a descrição do sistema genérico, é apresentada uma descrição dos protocolos de suporte à Comunicação em Grupo.

3.1 Modelo

O Appia é uma plataforma de suporte a composição de protocolos concretizada em JAVA, que suporta composições complexas. Posteriormente foram desenvolvidos diversos protocolos, nomeadamente um conjunto de protocolos de suporte à Comunicação em Grupo, assim como diversos protocolos para comunicação *ponto-a-ponto*.

Os componentes nucleares do Appia, na sua forma original, estão representados na Figura 3.1.

A composição de protocolos é dividida em dois níveis, um estático que define uma Qualidade de Serviço e um dinâmico que concretiza essa Qualidade de Serviço. No nível estático temos que cada protocolo é definido como uma *camada*, classe LAYER, sendo a sua correspondente dinâmica uma *sessão*, classe SESSION, que pode-se definir como sendo uma instância da camada. Uma QoS do Appia define uma pilha de cama-

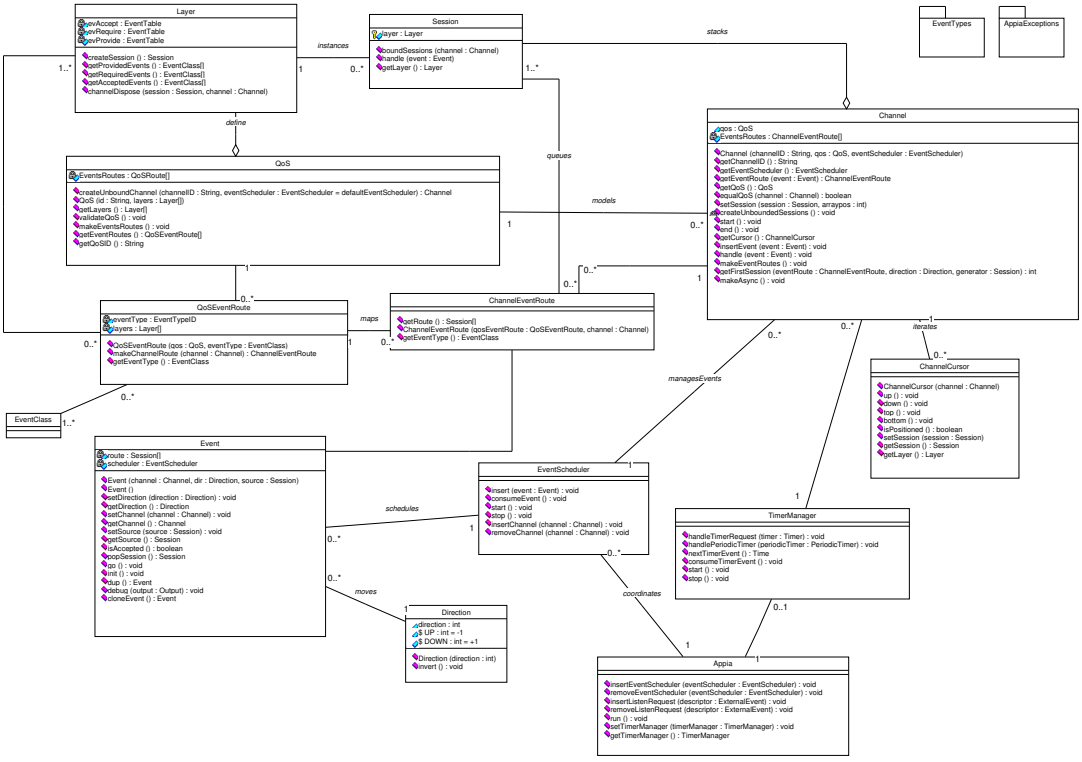

Figura 3.1: Representação original em *UML* do Appia

das, enquanto um *canal*, classe CHANNEL, define uma pilha de sessões, sendo portanto uma instanciação de um QOS.

O Appia é orientado aos eventos, sendo toda a interacção entre camadas e com o Appia em si feita por eventos. É oferecido um modelo aberto, ou seja, pode-se adicionar facilmente novos eventos. O mecanismo de herança é o instrumento principal para permitir a correcta utilização dos eventos e para que se possa conseguir um alto nível de reutilização. Todos os eventos têm que descender da classe EVENT. Sucessivas versões duma camada podem ir enriquecendo os atributos dos eventos gerados, criando novos eventos que descendem dos anteriores, mantendo assim a compatibilidade. Cada evento é caracterizado por três atributos: o canal que vai percorrer, a direcção (ascendente ou descendente) que vai tomar e a sessão que o gerou. Só quando estes três atributos são conhecidos é que um evento pode ser enviado.

Uma camada define igualmente o conjunto dos eventos que gera, que precisa e

que aceita. Com estes três conjuntos podemos efectuar uma série de operações: pode-se efectuar uma validação básica da pilha, ao verificar se todos os eventos necessários a qualquer das camadas, são gerados por alguma outra camada; e pode-se também optimizar o caminho percorrido pelos eventos.

Em vez de um evento percorrer todas as camadas da pilha, como noutros sistemas, no Appia um evento só percorre as camadas que indicaram que aceitam o evento. Testes demonstraram que esta simples optimização pode significar uma melhoria significativa (Miranda, 2001). Estes caminhos são igualmente definidos em dois níveis: o do *QoS*, como QoSEventRoute, que corresponde às camadas que aceitam o evento; e o do canal, como ChannelEventRoute, que define as sessões que o evento vai realmente percorrer. Estes caminhos podem ser calculados no inicio de operação do sistema, por isso a sua determinação não penaliza o funcionamento normal.

A flexibilidade de composição do Appia advém da capacidade de se poder atribuir a mesma sessão a vários canais, podendo assim gerar composições como a exibida na Figura 3.2.

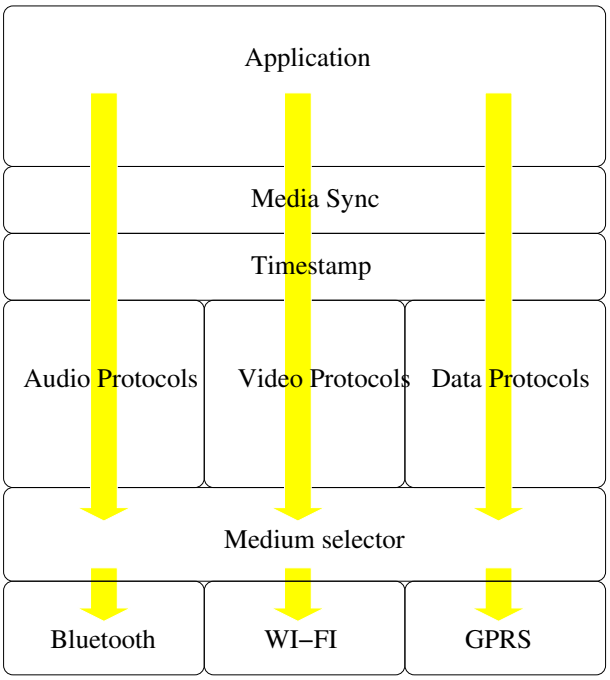

Figura 3.2: Exemplo de composição com sessões partilhadas por vários canais

Esta utilização da mesma sessão em vários canais só pode ser alcançada de forma explicita, ou seja, tem que ser o programador a explicitamente colocar nos vários canais

a sessão partilhada. Mas nem sempre essa partilha é necessária, podendo nesse caso a atribuição das sessões de um canal ser implícita. Além destes dois casos, pode ser útil serem as sessões a definirem elas próprias quais serão as outras sessões. Estes três mecanismos de atribuição de sessões aos canais são suportados da seguinte forma:

1. Quando um canal é criado a partir de um QoS não tem sessões atribuídas, só tem o espaço para elas. Quem criou o canal pode colocar explicitamente sessões nesses espaços antes de o iniciar, desde que as sessões sejam instâncias das camadas correspondentes na QoS.

2. Quando o canal é iniciado, o núcleo percorre o canal e pede a todas as sessões colocadas explicitamente que preencham os espaços livres. Isto é feito pela invocação do método boundSessions das sessões.

3. Se após isso ainda permanecerem espaços livres, então o núcleo preenche-os implicitamente, pedindo às camadas correspondentes que criem novas sessões.

Estas operações requerem um manuseamento da pilha do canal, que é feito através da classe CHANNELCURSOR.

Cada canal tem associado um escalonador, EVENTSCHEDULER, que é o responsável pela entrega dos eventos às sessões, seguindo os caminhos definidos para cada evento. Esta entrega requer algum cuidado pois deve respeitar uma ordenação FIFO, por forma a ser intuitivo e facilitar o desenvolvimento dos protocolos. Isto significa que se uma sessão s_1 recebe e envia o evento e_1 e depois recebe e envia o evento e_2 na mesma direcção, então a camada imediatamente a seguir nessa direcção deverá receber e_1 seguido de e_2. Mesmo que a sessão seguinte não pertença ao caminho de qualquer dos dois eventos, conceptualmente a regra deve-se aplicar.

A entrega dos eventos às sessões é feita por um único ponto, pelo método handle de cada sessão. Certos eventos são também entregues ao canal, pelo seu método handle. Estes eventos são chamados de eventos de canal, descendentes de CHANNELEVENT. Os mais importantes são os associados com temporizadores e o ECHOEVENT. Este permite que uma camada envie um evento que percorrerá o canal todo, em vez de só a parte que vai desde a sessão geradora até a um dos extremos do canal. Isto é conseguido pelo

facto de o ECHOEVENT transportar um outro evento, que será colocado no canal quando ele chegar a uma das extremidades do mesmo. O evento é enviado na direcção oposta à que o ECHOEVENT seguia.

Os temporizadores periódicos ou não, são controlados pelo núcleo, mas o seu pedido e as notificações dos mesmos é feita através de eventos, respectivamente PERIODICTIMER e TIMER. A gestão destes temporizadores é feita pela classe TIMERMANAGER.

Para a comunicação entre processos também são usados eventos, neste caso descendentes de SENDABLEEVENT. Mas o conteúdo da mensagem realmente enviada pela infraestrutura de rede, não é o evento mas sim um atributo deste, a MESSAGE, ou seja, qualquer informação que se pretenda enviar para outro processo via Appia deve ser colocada na MESSAGE de um SENDABLEEVENT. O envio da mensagem pelo meio de comunicação usado não é executado pelo núcleo, ficando a cargo de uma ou mais camadas. Estas conhecem o destino da mensagem pelo atributo dest de SENDABLEEVENT. Análogamente no destino as sessões sabem a origem pelo atributo source. De notar que o formato da identificação do destino e da origem pode variar enquanto o evento percorre o canal.

Resumindo, existem dois mecanismos para troca de informação entre camadas. Para camadas que se encontrem na mesma pilha temos os atributos públicos dos eventos, enquanto que para camadas que se encontrem em pilha diferentes temos a MESSAGE.

3.2 Concretização

A concretização seguiu fielmente a abrangente definição contida em (Miranda, 2001), pelo que a maior parte da concretização foi simples. Existiram no entanto alguns aspectos que necessitaram de tratamento e concretização cuidada, que serão apresentados de seguida.

3.2.1 Motor de Eventos

A concretização do motor de eventos requereu algum cuidado devido à necessidade de garantir uma ordenação FIFO dos eventos numa determinada direcção.

À primeira vista pode não aparentar uma grande dificuldade, mas se tivermos em linha de conta que os eventos não percorrem todas as camadas, mas a ordenação deve ser mantida como se percorressem, então temos pela frente algo mais complexo. Além disso, o motor funciona intensamente, pois como o sistema é orientado aos eventos tudo gira em torno destes. Isto implica que para o sistema ter um desempenho razoável o motor de eventos deve ser o mais eficiente possível, consumindo o menos possível dos recursos. O algoritmo concretizado, que prima pela simplicidade, baseia-se em para cada evento esgotar a sua relação de causalidade, ou seja, é escolhido um evento e percorrido todo o caminho desse evento, bem como os caminhos de todos os eventos causados por ele.

No entanto, mesmo dentro destes eventos, um cuidado especial deve ser tomado no que diz respeito à direcção que os eventos tomam. Uma aproximação simplista que ignora a direcção pode levar à violação da regra FIFO, o que aliás aconteceu nas primeiras concretizações. Por exemplo, vamos assumir uma concretização em que o próximo evento a ser entregue é o primeiro enviado pela camada actual, independentemente da direcção. Suponhamos que uma camada recebe o evento e_1, com direcção descendente, e a seguir gera e envia para cima um evento e_2 antes de reenviar e_1. Como e_2 foi o primeiro evento a ser enviado pela camada é o próximo a ser entregue. A camada que o recebeu reenvia-o só que para baixo, pelo que voltará à camada que o gerou. Nesta podemos dizer que, no sentido descendente, viu os eventos e_1 e e_2 por esta ordem. Quando a camada reenvia e_2 este será entregue então à camada inferior. Só depois é que e_1 será entregue, pelo que nesta camada a ordem pela qual os eventos foram observados foi e_2 seguido de e_1. A solução para este problema passa por escolher primeiro os eventos enviados na direcção do evento que os originou. Só após todos os eventos gerados nessa direcção terem sido entregues é que se processam os eventos enviados na outra direcção. O processo vai-se repetindo, até não existirem mais eventos a processar.

Este algoritmo apresenta algumas deficiências, principalmente a que uma incorrecta concretização de duas sessões, pode levar a um cenário em que as duas sessões trocam sucessivamente eventos entre elas, impedindo que outros eventos sejam processados, provocando na prática o bloqueio do sistema. No entanto, a sua simplicidade significa que o tempo gasto no escalonador é mínimo e que este gasta poucos recursos, e foi o único algoritmo testado que satisfez as condições necessárias. Além disso uma análise ao ambiente normal em que é utilizado, ou seja na comunicação entre processos, revelou que na maioria das vezes os eventos associados a essa comunicação percorrem todo o seu caminho, gerando poucos ou nenhuns eventos, e saem do sistema.

Seguindo as indicações dadas pelo trabalho anterior no Ensemble, que tinha revelado que o *reciclador automático de memória* têm um efeito importante sobre o desempenho do sistema, as estruturas internas do escalonador são geridas explicitamente. Foi criada uma colecção de objectos correspondentes às estruturas internas, e é desta colecção que são retirados os objectos necessários. O tamanho desta colecção é dinâmico, crescendo e diminuindo à medida das necessidades do sistema.

3.2.2 Actividades

Na concepção do Appia decidiu-se que este executaria no âmbito de uma única actividade ("thread"), tentando assim evitar a complexidade inerente à sincronização de actividades.

A linguagem de programação usada, o JAVA, não oferece, pelo menos até à versão 1.4.1, um mecanismo que permita esperar pela recepção de dados em vários descritores, por um período de tempo máximo. Estes descritores podem corresponder aos pontos de recepção de mensagens vindas da rede, *sockets*, ou à recepção de dados dos dispositivos de entrada, por exemplo teclado. Para efectuarmos a espera pela recepção dos dados resta-nos unicamente a invocação da chamada ao sistema que efectua essa recepção, que é por norma bloqueante, o que significa que se a invocássemos no âmbito da actividade do Appia suspenderiamos todo o sistema. Para evitar isto temos que ter uma actividade para cada descritor onde estamos à espera, que se bloqueia na chamada ao sistema que permite efectuar a recepção. Após a recepção dos dados é envi-

ado um evento no canal adequado com esse dados. Os dados recebidos da rede são colocados na MESSAGE do SENDABLE EVENT correspondente.

Estes eventos, pela sua natureza, são colocados no canal pelos mecanismos de envio de eventos de forma asincrona, concretizados pelo método asyncGo de cada evento. Este método, que não constava do desenho original do Appia, difere do método usual go no facto de não conter uma sessão originária e efectuar certas operações de sincronização de actividades. O método asyncGo efectua igualmente certas validações, para impedir por exemplo que seja invocado pela actividade principal do Appia.

Para além disso o próprio núcleo necessita de mecanismos de sincronização de actividades, para garantir a coerência entre a actividade que executa o núcleo e todas as outras referidas.

De notar que a interacção entre estas actividades e a sessão que a criou, pode requerer igualmente alguma forma de sincronização, mas esta é da responsabilidade de quem concretizou a camada. Este deve ter sempre em conta que nunca deve bloquear a actividade do núcleo, pois isso significaria bloquear todo o sistema.

3.2.3 Gestor de Temporizadores

O função do Gestor de Temporizadores é receber os eventos a pedir um temporizador, que pode ser periódico ou não, e quando o período de tempo pedido passar reenviar o evento, ou uma copia no caso dos periódicos. O evento é enviado no canal em que foi recebido, mas no sentido inverso.

Para evitar o recurso constante a chamadas ao sistema operativo para controlar o tempo, o Gestor concretiza um pseudo-relógio. Este pseudo-relógio é periodicamente sincronizado com o relógio do sistema. A concretização do pseudo-relógio só é possível através de um certo controlo do tempo gasto na execução, o que só é minimamente possível através do uso de uma actividade diferente da do núcleo.

Devido a esta concretização do Gestor, os eventos associados aos temporizadores são enviados de volta ao canal, pelos mecanismos para envio de eventos de forma asincrona.

3.2.4 Message

Seguindo direcções tomadas no desenvolvimento do sistema Ensemble, no desenvolvimento do Appia foram tomadas diversas medidas no sentido de controlar de forma eficaz o conteúdo das mensagens que são trocados entre os processos. Devido ao seu tamanho potencial e ao facto de se usar um reciclador automático de memória, estas medidas visam essencialmente efectuar uma gestão explicita das estruturas de dados usadas, com a máxima reutilização possível das estruturas já alocadas.

Este trabalho de optimização centrou-se obviamente na classe MESSAGE, a classe correspondente ao conteúdo das mensagens trocadas entre processos. Aquando da especificação da classe (Miranda, 2001) as necessidades de reutilização e optimização já tinham sido levadas em conta e o interface foi baseado no do x-Kernel (Hutchinson & Peterson, 1991). A principal particularidade reside no facto de que quando se pretende colocar informação na MESSAGE o método para o fazer não coloca directamente a informação, em vez disso devolve um bloco de dados onde essa informação pode ser colocada.

A principal característica da concretização da MESSAGE efectuada reside no facto de os dados não estarem num bloco único continuo (um *byte[]* em JAVA) até à chamada ao sistema operativo que efectua envio pelo meio de comunicação. Durante o trajecto pelo canal os dados vão sendo colocados numa fila de blocos de dados, com novos blocos sendo adicionados à fila sempre que necessário. Isto evita sucessivas reservas de blocos consecutivamente maiores e evita igualmente as cópias necessárias para manter os dados contínuos.

Apesar dos cuidados na sua concretização, a classe MESSAGE não oferece ferramentas que permitam a colocação de tipos de dados mais complexos. Tudo tem que estar na forma um vector de octetos (*byte []*), ficando o programador com a necessidade de converter os dados de formatos mais complexos para esta representação. Com o intuito de auxiliar o desenvolvimento dos protocolos foi criada uma nova classe, a EXTENDEDMESSAGE, que oferece exactamente essas ferramentas de conversão para os tipos básicos do JAVA.

O desenvolvimento da EXTENDEDMESSAGE foi lento e demorado, sobretudo devido a problemas de concepção que surgiram da tentativa de usar os mecanismos do JAVA que permitem codificar objectos num vector de octetos (*byte []*). O principal problema prendia-se com a diferença no paradigma de acesso entre a MESSAGE e os "streams" JAVA usados para codificar os objectos. O acesso à MESSAGE assume uma ordenação LIFO enquanto os "streams" do JAVA usam uma ordenação FIFO. Compatibilizar essa diferença de forma eficiente é complicado. Além disso, a codificação dos objectos efectuada pelo JAVA adiciona diversa informação que não é normalmente necessária, tendo em conta o uso dado pelas sessões do Appia. Esta informação desnecessária vai afectar o tamanho dos cabeçalhos colocados pelo sistema, afectando dessa forma o desempenho final.

A solução para estes problemas passou pela concretização de novos mecanismos que permitem codificar num vector de octetos, os objectos das classes mais vezes colocadas nas mensagens. Estes mecanismos, correspondentes a um método para codificar e outro para descodificar, fazem parte da própria classe a codificar, ou descodificar, e não da EXTENDEDMESSAGE, embora usem funcionalidade oferecida por esta, nomeadamente a que permite codificar e descodificar os tipos básico do JAVA.

Apesar de ter um problema de desempenho a capacidade de adicionar um objecto genérico a uma EXTENDEDMESSAGE é útil, por permitir um desenvolvimento mais rápido das camadas. Por esta razão esta capacidade foi também colocada. Desta forma para as classes menos usadas não é essencial a concretização dos mecanismos de codificação referidos.

3.3 Comunicação em Grupo

Após a concretização do núcleo do Appia foi desenvolvido o suporte para Comunicação em Grupo. Isto resultou na concretização de várias camadas, que executam diversos protocolos. De seguida serão apresentados o conceito, o modelo geral da concretização efectuada e depois serão descritas os eventos e as funcionalidades principais.

3.3.1 Conceito

O sistema de Comunicação em Grupo concretizado oferece *sincronia na vista*. A *sincronia na vista* é um modelo que disponibiliza, a todos os membros do grupo, informação coerente sobre os membros que constituem o grupo. Esta informação é chamada de *vista*. O modelo garante que todos os membros activos recebem todas as vistas e que estas são entregues na mesma ordem em todos eles. É igualmente garantido que qualquer mensagem enviada no contexto de uma determinada vista é entregue, pelo menos, a todos os membros da vista actual que farão parte da vista seguinte. Na concretização efectuada é garantido que a mensagem é entregue antes da vista seguinte. Deste modo todas as mensagens podem ser associadas com a vista na qual foram enviadas, porque existe a garantia que serão entregues antes de próxima vista. Para impor esta garantia é necessário que seja executado um processo de escoamento, que garanta que nenhum membro envie novas mensagens durante o processo de mudança de vista. Se tal acontecesse, seria impossível garantir que todos os membros já tivessem entregue todas as mensagens da vista.

Em redes de larga escala a probabilidade de ocorrência de partições da rede (o conceito de partição da rede é descrito de forma sumária na Secção 2.2.1.2) é elevada. Nesses cenários é possível que existam duas vistas do mesmo grupo a operar de forma concorrente em partições separadas. Quando as partições são reunificadas é necessário unir as vistas concorrentes do grupo. Isto implica um processo de sincronização entre as duas vistas, de forma a preservar a semântica da *sincronia na vista* e eventualmente entregar uma vista com todos os membros do grupo.

3.3.2 Modelo

O trabalho inicial que levou ao desenvolvimento do Appia foi feito no sistema Ensemble, dai que os protocolos de suporte à Comunicação em Grupo tivessem sido fortemente influenciados pelos oferecidos por esse sistema. No entanto, foram efectuadas diversas modificações numa tentativa de os tornar mais independentes.

No Ensemble, como todo o sistema foi desenvolvido para a Comunicação em

Grupo, as camadas foram desenvolvidas só para esse paradigma. No Appia, embora a Comunicação em Grupo seja um dos paradigmas mais usados, o sistema foi desenvolvido para suportar também comunicação *ponto-a-ponto*. Assim, na concretização efectuada os protocolos de suporte a Comunicação em Grupo assumem que existe suporte a comunicação *ponto-a-ponto* fiável, com ordenação FIFO das mensagens, para simplificar a sua concretização. Estas propriedades de fiabilidade e ordenação, são oferecidas por camadas que se encontram abaixo das de Comunicação em Grupo e que foram desenvolvidas independentemente destas.

Embora tenha havido uma tentativa de modularização dos diversos protocolos, alguns assumem a existência de outros num posicionamento relativo definido. Por exemplo, o protocolo executado pela camada *Inter* assume a existência da camada *Intra* e que esta se encontra abaixo dele. Uma explicação destes dois protocolos será apresentada mais à frente neste capitulo.

Cada grupo é identificado pela classe GROUP, cada vista é identificada pela classe VIEWID e cada membro do grupo é identificado pela classe ENDPT. Cada vista tem um coordenador que é conhecido através do identificador da vista. Normalmente o coordenador é o membro de índice zero, ou seja, o mais antigo.

A instalação de uma nova vista é sinalizada pelo evento VIEW. Este evento percorre todo o canal no sentido ascendente. Parte do seu conteúdo, um dos seus atributos, é um objecto da classe VIEWSTATE. A classe VIEWSTATE contém informação geral relativa à nova vista, como a identificação da mesma ou os membros que a compõem. Esta informação é idêntica em todos os membros da nova vista. Além da VIEWSTATE é entregue também um objecto da classe LOCALSTATE, com informação local relativa à vista, como o índice do membro na vista. Esta informação é diferente para cada um dos membros da vista. Este esquema, e até os próprios nomes, é idêntico ao utilizado no Ensemble. No entanto, os atributos de cada uma das classes diferem dos do Ensemble.

A pilha normal de suporte à Comunicação em Grupo é apresentada na Figura 3.3. De salientar a existência de dois canais auxiliares, um usado para comunicar com o servidor de "gossip" e outro para a sincronização necessária à união de vistas concorrentes do mesmo grupo. A utilidade destes canais será descrita nas Secções 3.3.4.6 e

3.3.4.7 respectivamente. Ambos partilham a sessão de ligação ao meio de comunicação ("UdpSimple") com o canal principal. Mas enquanto que o canal para união de vistas também partilha a camada que garante comunicação fiável ponto-a-ponto com ordenação FIFO, o canal de comunicação com servidor de "gossip" utiliza uma camada diferente, porque não necessita de fiabilidade. Neste canal só é necessário ordenação FIFO e detecção de falha ponto-a-ponto.

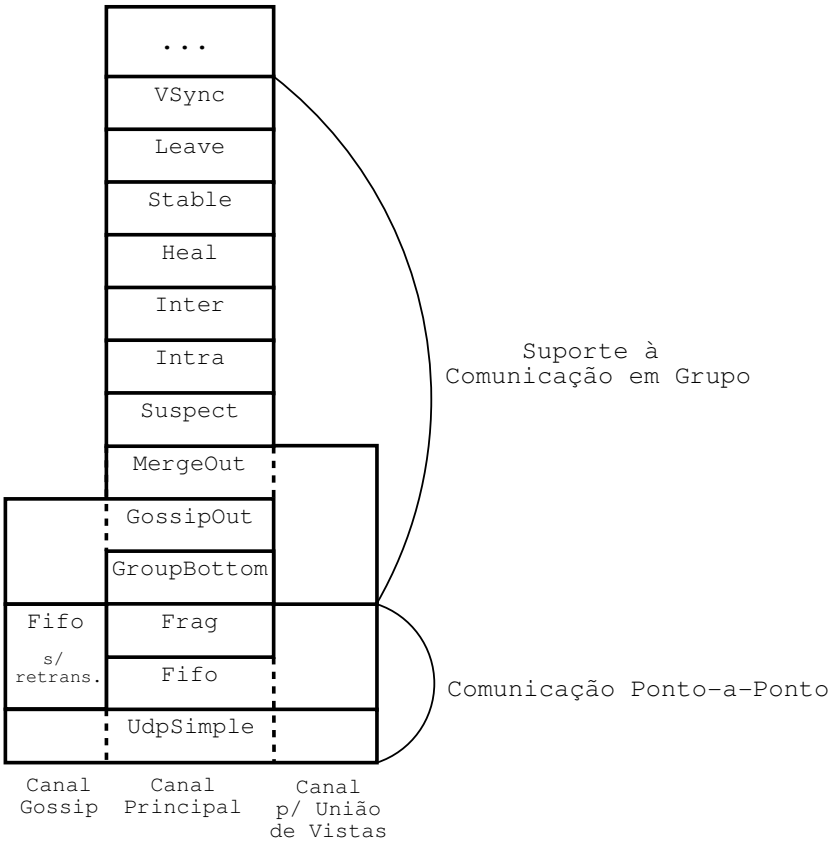

Figura 3.3: Composição usual com suporte à Comunicação em Grupo do Appia

3.3.3 Eventos

Todos os eventos relacionados com Comunicação em Grupo descendem de dois eventos base. Os eventos utilizados só para a comunicação entre as várias sessões de Comunicação em Grupo de um canal devem descender de GROUPEVENT. Um exemplo é o evento VIEW referido anteriormente.

Os eventos utilizados na comunicação entre os membros de um grupo devem des-

cender de GROUPSENDABLEEVENT. Como estes se destinam a comunicação entre processos, portanto a serem enviados pelo meio de comunicação usado, a classe GROUPSENDABLEEVENT descende de SENDABLEEVENT. Os eventos que descendem unicamente de GROUPSENDABLEEVENT destinam-se à comunicação com todo o grupo e são enviados para todos os membros do grupo. O seu atributo dest é ignorado. No entanto, a possibilidade de enviar para um subconjunto dos membros do grupo pode ser útil. Esta é oferecida descendendo o evento a enviar de SEND, além de ter que descender de GROUPSENDABLEEVENT. Para enviar estes eventos deve ser colocado no atributo dest os índices na vista corrente dos membros aos quais deve ser entregue. Quer descenda ou não de SEND, o atributo source só é valido no membro que recebe o evento, contendo o identificador, o ENDPT, do membro que enviou o evento. Como os membros são muitas vezes referenciados pelo seu índice na vista actual, para evitar que este tenha sempre que ser calculado a partir do identificador, no membro que recebeu o evento existe mais um atributo, o orig, que contém o índice do membro que enviou o evento.

Como referido anteriormente, um evento destinado à comunicação entre processos não é enviado todo pela rede, só a mensagem contida nele. Esta é representada pela classe MESSAGE. A Comunicação em Grupo foi desenvolvida para utilizar a versão extendida desta, que permite manipular tipos de dados mais complexos, a EXTENDEDMESSAGE. Assim todos os GROUPSENDABLEEVENT devem conter um EXTENDEDMESSAGE[1]

3.3.4 Funcionalidades

3.3.4.1 Ligação entre a Comunicação em Grupo e as camadas inferiores

A ligação entre a Comunicação em Grupo e as camadas inferiores, que oferecem comunicação *ponto-a-ponto*, é efectuada pela camada *GroupBottom*. Esta camada como o próprio nome indica deve ser a que se encontra mais abaixo das de suporte à Comunicação em Grupo.

Se as camadas inferiores suportarem a comunicação *ponto-a-ponto* de forma estrita,

[1]Na realidade contém uma OBJECTSMESSAGE, o primeiro nome dado a esta classe e que se mantém por motivos de retro-compatibilidade.

então cada evento enviado terá um só destino. Isso implica que, para cada evento destinado a todo o grupo seja criado um evento para cada um dos membros. Esse trabalho de gerar várias copias, uma para cada membro, do evento a enviar é uma das principais responsabilidades da camada. Com o objectivo de reduzir esta multiplicação dos eventos gerados foi criado um mecanismo diferente de identificação dos destinatários. A classe APPIAMULTICAST permite a identificação de um vector de destinatários. Se as camadas inferiores suportarem este mecanismo reduz-se o número de eventos gerados na pilha. Cabe à *GroupBottom* a análise do canal usado para determinar se deve usar ou não o mecanismo de APPIAMULTICAST. Actualmente diversas camadas suportam o mecanismo de APPIAMULTICAST.

Outra tarefa executada é a filtragem de todos os eventos relacionados com Comunicação em Grupo que não se destinam ao grupo e vista corrente.

3.3.4.2 Detecção de falhas

A detecção de falhas, mais correctamente a suspeição de falhas, é efectuada pela camada *Suspect*. O seu funcionamento é simples, baseando-se em parte na detecção de falhas oferecida pelos protocolos de comunicação sobre os quais o suporte à Comunicação em Grupo foi desenvolvido. Além desta, a camada efectua uma detecção de mais alto nível, igualmente simples, que se baseia no principio de que todos os membros não podem estar mais do que um determinado período de tempo sem enviar qualquer mensagem. Se as camadas superiores não enviarem por sua iniciativa, é enviada uma especificamente para esse fim. Assim, se um membro não receber qualquer mensagem de um membro durante um período de tempo estabelecido, assume-se que o membro falhou.

Sempre que uma falha, na realidade a suspeita de uma falha, acontece essa informação é transmitida aos outros membros do grupo e uma mudança de vista é iniciada.

3.3.4.3 Mudança de vista

A mudança de vista é controlada pela camada *Intra*. O processo de mudança de vista é despoletado de duas formas:

- Falha de um membro, sinalizada pelo detector de falhas (camada *Suspect*) através do evento *Fail*;

- Pedido por uma camada que se encontra acima da *Intra*, através do evento *ViewChange*.

Depois de iniciado, o processo de mudança de vista passa por três fases, que decorrem unicamente no coordenador actual da vista, que é o membro com índice mais baixo que se encontra activo:

Garantir a correcção da mudança. É enviado um evento NEWVIEW que percorre a pilha desde o topo no sentido descendente. Qualquer camada que pretenda garantir a correcção da mudança de vista (por exemplo a que garante a *sincronia na vista*) deve reter o evento até que o seu protocolo tenha terminado.

Definição da nova vista. Após o regresso do NEWVIEW é enviado um evento PREVIEW com uma proposta para a constituição da nova vista. Este evento percorre a pilha igualmente desde o topo no sentido descendente. Qualquer camada que pretenda alterar a composição da nova vista (por exemplo para juntar novos membros) deve alterar a proposta recebida no PREVIEW e reenviar o evento.

Entrega da nova vista. Após o regresso do PREVIEW a vista contida neste é enviada para todos os membros da vista actual. Depois de enviar a nova vista para os restantes membros, ou após a recepção dessa vista, é enviado um evento VIEW na pilha local com a nova vista, evento este que percorre a pilha desde o fundo em sentido ascendente.

Existe mais um motivo para uma vista ser entregue, a vista inicial. A vista inicial e obtida através do evento *GroupInit* enviado pela aplicação. Essa vista é entregue exactamente como recebida e não é propagada para ninguém.

3.3.4.4 Difusão fiável

A camada *Stable* tem como objectivo garantir a difusão fiável das mensagens de grupo, ou seja, qualquer mensagem destinada ao grupo e recebida por um dos membros correctos é recebida por todos os membros correctos. Para isto, todos os membros guardam as mensagens de grupo, de forma a poderem retransmitir as mesmas no caso de algum dos membros não as ter recebido e o emissor original não o poder fazer porque falhou.

Como o suporte a Comunicação em Grupo assume que existe comunicação fiável ponto-a-ponto, a *Stable* não necessita de guardar as mensagens enviadas pelo seu membro, só as recebidas dos outros membros. Periodicamente todos os membros enviam para os outros uma indicação das mensagens que receberam, de forma a que estes possam esquecer aquelas que já foram recebidas por todos os membros. A estas mensagens dá-se o nome de *estabilizadas*.

3.3.4.5 Sincronia na Vista

Garantir que a mudança de vista respeita a *sincronia na vista* é a responsabilidade da camada *VSync*. Durante a operação normal do grupo a camada efectua unicamente uma contagem das mensagens enviadas e recebidas de cada membro. Quando é recebido um evento NEWVIEW o protocolo de sincronização começa a funcionar.

Como é assumido que as mensagens são recebidas com uma ordenação FIFO ponto-a-ponto, não é necessário identificar todas as mensagens enviadas e recebidas, uma contagem é suficiente. Assim, o protocolo de sincronização resume-se a contar o número de mensagens enviadas e recebidas por cada membro, terminando quando as contagens forem iguais em todos os membros. Para que haja a garantia que o protocolo termina é necessário que os membros deixem de enviar mensagens. Para isso, quando o protocolo inicia a operação todos os membros enviam um evento BLOCKOK, que percorre a pilha desde o topo em sentido descendente. Qualquer camada que receba este evento pode reter o evento para enviar mensagens essenciais, mas após ter reenviado o evento não pode enviar mais mensagens.

Quando todos os membros receberam as mesmas mensagens o NEWVIEW é reenviado e o protocolo termina. De notar que a retransmissão de mensagens perdidas não é da responsabilidade desta camada, mas sim da de suporte à comunicação ponto-a-ponto fiável e da camada *Stable*.

3.3.4.6 Detecção de vistas concorrentes

A detecção de vistas concorrentes do grupo, ou seja, de outros membros do grupo que ainda não estão na mesma vista do grupo, é efectuada pela camada *Heal*. Isto é alcançado de duas formas diferentes consoante existe, ou não, o suporte a mecanismos de difusão selectiva ("multicast") por parte do meio de comunicação utilizado. Se esse suporte existir então as mensagens de cada uma das vistas concorrentes serão recebidas pelos membros da outra. Estas são detectadas primeiramente pela camada *GroupBottom*, que notifica a *Heal*, que depois inicia o processo de reunificação.

Se o suporte a difusão selectiva não existir então a detecção é feita através de um servidor de "gossip". Este é um servidor extremamente simples, que se resume a enviar uma mensagem recebida para todos os processos dos quais recebeu mensagens anteriormente. É garantido assim uma forma de difusão global. Periodicamente, a camada *Heal* do coordenador do grupo envia uma mensagem para o servidor de "gossip", com informação sobre a sua vista, como forma de anunciar a sua presença. Se o coordenador de outra vista do mesmo grupo receber este anuncio, sabe que existe uma vista concorrente e pode iniciar o processo de reunificação.

A comunicação com o servidor de "gossip" é feita através de uma canal diferente do usado para a comunicação do grupo, porque essa comunicação tem necessidades diferentes, por exemplo não é necessária a garantia de entrega. Este canal é criado e controlado pela camada *GossipOut*, que após a criação do canal se limita a executar o encaminhamento entre os dois canais.

3.3.4.7 União de vistas concorrentes

O processo de união de duas vistas concorrentes do mesmo grupo é efectuada pela camada *Inter*. O processo de união inicia-se quando a existência de uma vista concorrente é anunciada pela camada *Heal*. O processo de união compreende cinco passos:

1. O coordenador da vista mais recente efectua um pedido ao coordenador da vista mais antiga, ao qual se dá o nome de coordenador da união, para esta se efectuar;
2. O coordenador da união pode aceitar ou recusar este pedido. Um motivo para recusar é já se encontrar num processo de reunificação. Se aceitar, ambos os coordenadores iniciam um processo de mudança de vista;
3. Quando for recebido o evento PREVIEW pelo coordenador da vista mais recente, a nova vista nele contida é enviada ao coordenador da união;
4. Quando o coordenador da união tiver recebido o evento PREVIEW da sua vista, bem como a nova vista do coordenador da vista mais recente, é gerada a nova vista unificada pela concatenação das duas vistas. A nova vista unificada é então enviada ao coordenador da vista mais recente.
5. Após a geração ou recepção da nova vista reunificada, esta é colocada no evento PREVIEW e este é reenviado. A nova vista será entregue em todos os membros.

A comunicação entre os coordenadores durante o processo de união utiliza um canal diferente do usado na comunicação do grupo, canal este que é criado e controlado pela camada *MergeOut*.

3.4 Sumário

Este capítulo começa por apresentar o modelo do Appia. Apesar do modelo ser muito completo ainda existiam muitas opções a serem tomadas para resolverem aspectos relativos à concretização. Os aspectos mais importantes, como o uso de actividades, são apresentados. Finalmente foram desenvolvidos um conjunto de protocolos

para a plataforma, nomeadamente os de suporte à Comunicação em Grupo, que são descritos em pormenor.

4

Concretização dos Grupos Ligeiros

Os Grupos Ligeiros foram concretizados em diversas plataformas. Tratou-se de um processo evolutivo, em que cada concretização num novo sistema mais recente ia retirando conclusões sobre o sistema e sobre o próprio serviço. Desta forma foram sendo retiradas indicações sobre as necessidades que o serviço impõem às plataformas sobre as quais era concretizado, que foram utilizadas, em certos casos, para a melhoria/desenvolvimento das próprias plataformas. Além disso, o próprio serviço sofreu evoluções resultantes da própria experiência obtida nessas concretizações.

A apresentação seguirá a ordem cronológica pela qual as concretizações foram efectuadas. Assim começará com a concretização no Horus, da qual só será apresentada uma breve descrição, apresentando depois em maior detalhe as duas concretizações seguintes, no *Ensemble* e finalmente no Appia.

4.1 Grupos Ligeiros no Horus

A primeira versão dos Grupos Ligeiros (Guo & Rodrigues, 1997) foi concretizada na plataforma *Horus*. Esta primeira versão não tolerava partições da rede. A concretização traduziu-se no desenvolvimento de uma camada para o Horus, que executava a ligação entre uma "sub-pilha" que se encontrava por baixo, que representava o *grupo pesado*, e várias "sub-pilhas" que se encontravam por cima, que representavam os *grupos ligeiros* projectados nesse *grupo pesado*. As propriedades oferecidas pela "sub-pilha" do *grupo pesado* correspondem aos recursos partilhados pelos *grupos ligeiros* projectados. Já as "sub-pilhas" dos *grupos ligeiros* correspondem às propriedades únicas

de cada um destes. A principal função da camada era efectuar o encaminhamento dos eventos de uma "sub-pilha" para outra. Efectuava igualmente a monitorização das filiações dos *grupos ligeiros* e dos *grupos pesados*, para poder mudar as projecções entre estes quando as existentes se tornassem menos eficazes.

O sistema utilizava um algoritmo de *esvaziamento por software* ("software-flush"), para efectuar uma mudança de vista de um *grupo ligeiro* sem ser necessário mudar a vista do *grupo pesado*, que mais tarde, aquando da concretização dos Grupos Ligeiros no Ensemble, se descobriu ter uma falha. Uma explicação do algoritmo, do problema e da solução será apresentada na Secção 4.2.

Uma limitação da concretização, para além de não suportar partições, prendia-se com o facto de só poder existir uma camada do serviço de Grupos Ligeiros para todas pilhas. Isto devia-se ao facto do estado da camada ser guardado de forma global, por forma a poder estar disponível às varias pilhas. Desta forma era impossível ter várias instâncias do serviço a correr simultaneamente.

A concretização seguiu de muito perto a definição do conceito definida em (Guo & Rodrigues, 1997) e serviu sobretudo para validar o conceito e demonstrar que o serviço poderia melhorar o desempenho de um serviço de Comunicação em Grupo.

4.2 Grupos Ligeiros no Ensemble

Após a concretização no Horus, foi efectuada a concretização no Ensemble (Pinto *et al.* , 2001).

4.2.1 Posicionamento

A interface de aplicação do Ensemble não está posicionado no topo da pilha de comunicação. Em vez disso está colocado ao lado da pilha. O serviço de Grupos Ligeiros no Ensemble foi concretizado como uma nova interface que funciona por cima da original (Figura 4.1). Isto resultou na existência de dois módulos de interface, com o

do serviço de Grupos Ligeiros comunicando com o do Ensemble em vez da aplicação, e esta comunicando com o serviço de Grupos Ligeiros em vez de com o original.

A ideia inicial era colocar o serviço de Grupos Ligeiros como uma nova camada, que seria adicionada a pilha de protocolos e cruzaria horizontalmente várias pilhas, ou seja, seria comum a várias pilhas. Esta tinha sido a concretização adoptada na primeira concretização dos Grupos Ligeiros, no sistema Horus. No entanto, a impossibilidade de no Ensemble uma camada pertencer a varias pilhas, já que estas têm que ser estritamente verticais e independentes, ditou a configuração concretizada. Esta sofre de várias limitações:

- esta posição fixa limita as alternativas de configuração;
- todos os *grupos ligeiros* projectados no mesmo *grupo pesado* têm que usar exactamente a mesma pilha de protocolos de comunicação;
- o serviço de Grupos Ligeiros não tem acesso a certos eventos que percorrem a pilha de protocolos mas que não são notificados à aplicação.

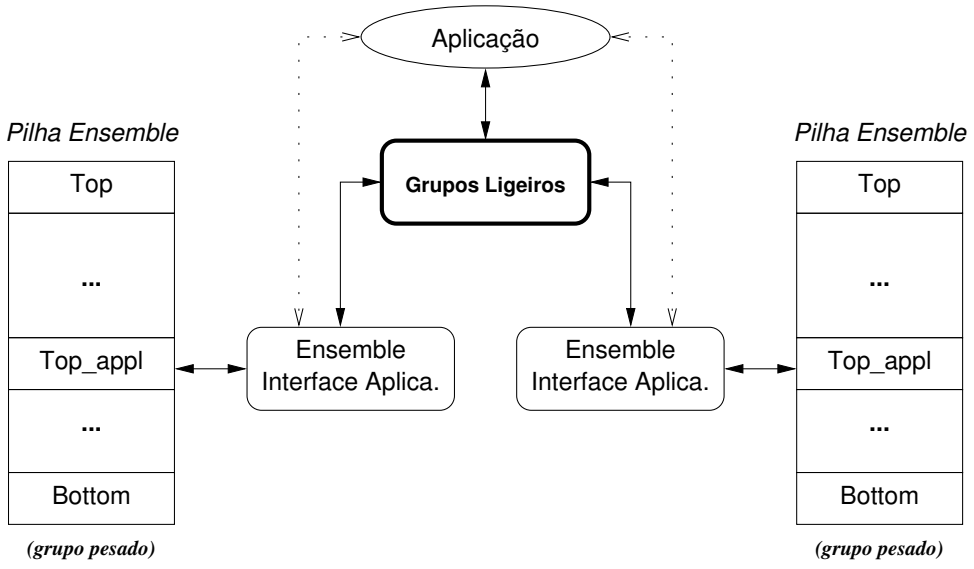

Figura 4.1: Posicionamento do serviço de Grupos Ligeiros no Ensemble

4.2.2 Funcionamento

Foram efectuadas algumas alterações ao conceito original (Rodrigues & Guo, 2000), numa tentativa de simplificação, adaptação ou optimização do desempenho.

Uma das alterações mais importantes e de maior impacto, foi a remoção do protocolo de *esvaziamento por software* do *grupo ligeiro*, que permitia efectuar uma mudança de vista do *grupo ligeiro*, respeitando o modelo de *sincronia na vista*, sem mudar a vista do *grupo pesado*. Este protocolo é impossível de ser realizado no Ensemble, porque quando este entrega uma vista é permitido o envio de mensagens imediatamente. Para manter a transparência os Grupos Ligeiros também teriam que permitir este envio. O problema ocorreria se durante o processo de *esvaziamento* ocorresse uma mudança de vista do *grupo pesado*. Esta mudança de vista seria precedida de uma notificação de que os membros do *grupo pesado* não poderiam enviar mensagens, que poderia não ser recebida em todos os membros pela mesma ordem. Isto poderia levar a que certos membros do *grupo ligeiro* entregassem a nova vista antes da notificação do *grupo pesado* e outros só depois. Estes últimos teriam que entregar a vista, para manter o estado coerente para as aplicações, e ao mesmo tempo não a poderiam entregar porque se o fizessem permitiam o envio de mensagens por parte da aplicação, algo que o *grupo pesado* não permitia. Com a impossibilidade de utilização do protocolo de *esvaziamento por software*, cada mudança de vista de um *grupo ligeiro* implica uma mudança de vista do *grupo pesado* onde se encontra projectado, com elevada interferência para os outros *grupos ligeiros* projectados.

Foi também relevante a decisão, seguindo a adoptada no Ensemble, de eliminar a primitiva "Join", não existindo nenhuma funcionalidade para juntar explicitamente um novo membro ao grupo. Em vez disso, existe a funcionalidade que permite juntar duas ou mais vistas, essencial para permitir a recuperação de partições da rede. Assim, juntar um membro ao grupo é realizado como a união de uma vista com um único elemento, o novo membro, à vista com os restantes membros do grupo. Ao unir estes dois mecanismos simplifica-se o sistema.

Outra alteração foi o uso de um modelo centralizado para a detecção e reunificação de vistas concorrentes de um *grupo ligeiro*, dentro de um *grupo pesado*. Vistas concor-

rentes pode ser definido como duas ou mais vistas disjuntas do mesmo *grupo ligeiro* que existem ao mesmo tempo. O conceito original, mas não concretizado, estipulava que as diferentes vistas do *grupo ligeiro* trocavam os respectivos estados e depois acordavam numa nova vista reunificada. Na concretização efectuada todas as vistas de *grupos ligeiros* informam o coordenador do *grupo pesado* dos seus estados, este detecta vistas concorrentes do mesmo *grupo ligeiro* e decide como será a nova vista reunificada. Esta informação é transmitida numa única mensagem para todo o *grupo pesado*. A escolha desta aproximação centralizada deveu-se a duas razões: *i*) as vistas a juntar e a constituição da nova vista, são dadas pelo coordenador do *grupo pesado*, não sendo necessário nenhum protocolo adicional para alcançar um acordo; *ii*) na ausência de suporte à difusão selectiva ("multicast") é reduzido o número de mensagens necessárias à detecção e correcção das vistas concorrentes.

Alguma da funcionalidade e serviços oferecidos pelo Ensemble não são suportados pelo serviço de Grupos Ligeiros, ou porque seria necessário duplicar inteiramente funcionalidade complexa já concretizada ao nível do *grupo pesado*, ou porque são totalmente incompatíveis com o conceito de Grupos Ligeiros. Dois exemplos são respectivamente as funcionalidades de "XferDone" e "migrate".

4.2.3 Reciclagem Automática de Memória

O facto do Ensemble ser concretizado em OCAML [1] tem um inconveniente que diz respeito ao uso de um reciclador automático de memória. Este tem um mau desempenho com variáveis de grande dimensão, como aquelas correspondentes aos dados das mensagens trocadas entre os membros de um grupo. Para garantir um bom desempenho o programador tem que gerir explicitamente estas variáveis, sendo que o Ensemble oferece um conjunto de ferramentas com esse propósito. Outro problema, relacionado também com o reciclador automático de memória, é o facto de só se poder utilizar as primitivas imperativas da linguagem, pois as primitivas funcionais (ex: listas) geram muitos objectos, que implica maior carga de trabalho do reciclador automático de memória e uma degradação do desempenho. Assim, embora o OCAML

[1] Já existe uma versão do Ensemble em C mas que é posterior ao trabalho efectuado.

seja uma linguagem principalmente funcional só se podem usar as suas primitivas imperativas, o que pode ser um pouco contra natura. Estas regras foram seguidas na concretização do serviço de Grupos Ligeiros.

4.2.4 Módulos

Na concretização do serviço de Grupos Ligeiros foi usada uma estrutura modular para os seus diferentes componentes. A Figura 4.2 apresenta uma representação da estrutura usada.

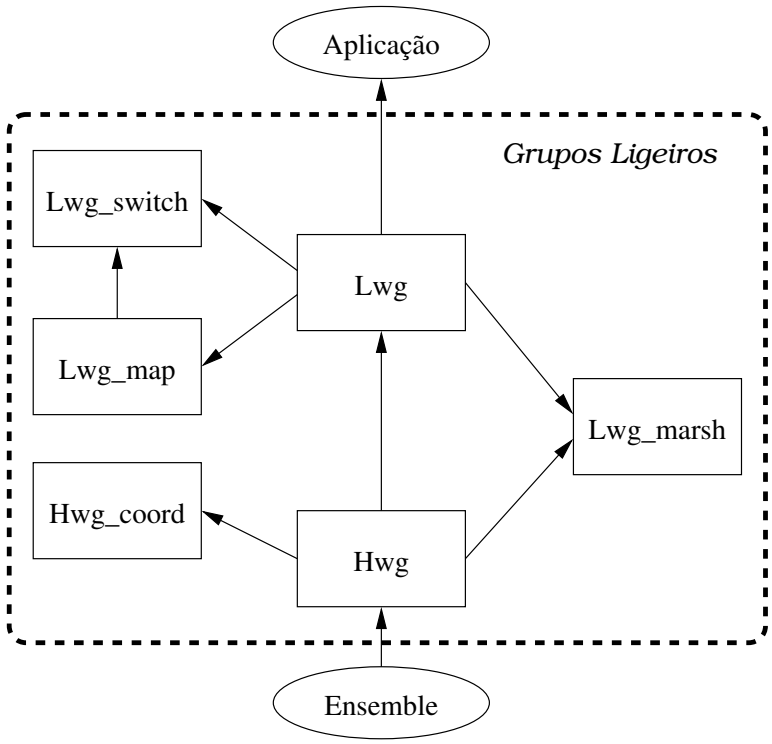

Figura 4.2: Estrutura do serviço de Grupos Ligeiros no Ensemble

Antes de descrever de forma mais detalhada os vários módulos é necessário fazer referência ao modo como os módulos comunicam entre si. Apesar de existir uma hierarquia lógica entre eles, não existe um mecanismo uniforme de comunicação, como acontece no próprio Ensemble, sendo a comunicação feita através de invocação directa de funções. O serviço em si é monolítico. Analisemos agora os módulos de baixo para cima.

O primeiro modulo é o *Hwg*, cuja principal função é a comunicação com a interface original do Ensemble, concretizando as funções de chamada ("call-back") e registando-as no Ensemble como se tratasse de uma aplicação. Outra função efectuada é a gestão da colecção de *grupos pesados*, sendo assim o responsável pela criação e remoção destes. Finalmente, é responsável pelo encaminhamento para o *grupo ligeiro* das notificações que recebe do Ensemble, tendo para isto que controlar as projecções locais entre *grupos ligeiros* e *grupos pesados*. Ou seja, quando uma mensagem chega do Ensemble é retirada informação que identifica o *grupo ligeiro* à qual pertence, que foi colocada pelo serviço de Grupos Ligeiros no emissor, sendo depois entregue unicamente a este *grupo ligeiro*. Se o membro do *grupo pesado* que recebe a mensagem não é membro do *grupo ligeiro* ao qual a mensagem pertence, ela é descartada. Definimos como projecções locais como todas as projecções que são conhecidos do processo.

Durante a operação dos Grupos Ligeiros pode-se chegar a um estado onde existem vistas concorrentes do mesmo *grupo ligeiro*. Para detecção e reunificação destas foi usada uma aproximação centralizada, o que significa que o coordenador do *grupo pesado* desempenha um papel especial. Esta funcionalidade é desempenhada pelo próximo módulo o *Hwg_coord*. As suas tarefas resumem-se a guardar as vistas dos *grupos ligeiros* projectados no *grupo pesado*, verificar se existem vistas concorrentes nestas, e se existirem, enviar uma mensagem a informar as vistas em causa que se devem reunificar e de qual deve ser a nova vista reunificada. Para saber as vistas existentes, após cada mudança da vista do *grupo pesado* todos os coordenadores de vistas dos *grupos ligeiros* projectados enviam as respectivas vistas ao coordenador do *grupo pesado*. Se o coordenador do *grupo pesado* detectar vistas concorrentes, envia a mensagem a pedir a reunificação e pede ao Ensemble uma nova vista do *grupo pesado*, que marcará o momento em que as novas vistas reunificadas devem ser entregues. Isto só pode ser feito pelo coordenador do *grupo pesado*, porque só o coordenador de um grupo no Ensemble pode pedir novas vistas. Para diminuir o número de mensagens, todas as notificações de reunificação de vistas concorrentes são agrupados na mesma mensagem, que será enviada para todos os membros do *grupo pesado*. Quando um membro de um *grupo ligeiro* deseja sair do grupo é necessário uma nova vista, pelo que o coordenador do *grupo pesado* deve ser informado disto, para que peça uma nova vista do *grupo pesado*

que marcará o momento da saída.

Outra função importante desempenhada pelo coordenador do *grupo pesado* é a comunicação com o *Servidor de Nomes*. Desta forma reduz-se o número de mensagens trocadas com o Servidor de Nomes, porque como o coordenador do *grupo pesado* sabe as vistas de todos os *grupos ligeiros* projectados no *grupo pesado*, pode numa única mensagem informar o Servidor de Nomes destas.

Apesar do coordenador do *grupo pesado* desempenhar várias funcionalidades no âmbito do serviço de Grupos Ligeiros, a complexidade destas não é grande, causando pouca interferência no seu funcionamento normal.

O próximo módulo é o responsável pela ligação com a aplicação. É chamado de *Lwg*. Como um dos requisitos do serviço de Grupos Ligeiros é a transparência, a interface oferecida pelo serviço é idêntica à oferecida pelo Ensemble. Isto implica que o serviço tem que guardar as funções registadas pela aplicação e tem que as invocar sempre que necessário. Normalmente, sempre que o Ensemble invoca uma das funções registadas pelo *Hwg*, a função similar registada pela aplicação no *Lwg* é invocada. Outra funcionalidade desempenhada pelo módulo é a de eliminar da vista do *grupo ligeiro* todos os membros que já não pertencem a vista do *grupo pesado*, ou seja, que falharam. Também é responsabilidade do módulo a recepção e entrega da vista reunificada do *grupo ligeiro* recebida do coordenador do *grupo pesado*. Isto significa que para determinar uma nova vista do *grupo ligeiro* não é necessário os seus membros trocarem qualquer mensagem específica. O módulo efectua igualmente a ligação com outra funcionalidade desempenhada por *grupo ligeiro*, nomeadamente, o protocolo de mudança de *grupo pesado* e a avaliação da adequação da projecção entre o *grupo ligeiro* e o *grupo pesado*. Estas duas funcionalidades são desempenhadas pelos dois próximos módulos.

O próximo modulo é responsável pela execução do protocolo que permite mudar um *grupo ligeiro* de um *grupo pesado* para outro, de forma correcta e transparente. Chama-se *Lwg_switch*. O seu funcionamento é simples, começando por informar todos os membros do *grupo ligeiro* para se juntarem ao *grupo pesado* de destino. Quando estes informarem que já se juntaram, é pedida uma nova vista do *grupo pesado* original

que marcará a mudança. Após isto, é entregue uma nova vista do *grupo ligeiro* já no *grupo pesado* de destino. Até à mudança de vista no *grupo pesado* original o grupo funciona normalmente, recebendo e enviando mensagens. O protocolo só é abortado se for recebida uma mensagem do coordenador do *grupo pesado* a pedir que duas vistas concorrentes do *grupo ligeiro* se reunifiquem. Isto é necessário para evitar que uma das vistas do *grupo ligeiro* fique incoerente.

A adequação da projecção entre um *grupo ligeiro* e um *grupo pesado* pode variar ao longo do tempo, como explicado em (Rodrigues & Guo, 2000), à medida que as respectivas vistas se vão modificando. Por exemplo, se o número de membros do *grupo pesado* vai aumentando mas o número de membros do *grupo ligeiro* se mantém, isto pode levar a um estado no qual os membros do *grupo ligeiro* são uma parte muito pequena do *grupo pesado*. Neste estado, o funcionamento do *grupo ligeiro* causará muita interferência. A solução é mudar o *grupo ligeiro* para um *grupo pesado* adequado. Uma projecção é considerada correcta se o *grupo ligeiro* não for uma *minoria* do *grupo pesado*. Dizemos que a vista v_1 é minoria da vista v_2 se (Rodrigues & Guo, 2000):

$$(\forall m \in v_1 : m \in v_2) \text{ e } (tamanho(v_1) \leq tamanho(v_2)/k)$$

Na concretização efectuada $k = 4$.

A monitorização da correcção da projecção é efectuada pelo módulo seguinte, o *Lwg_map*. A avaliação da correcção da projecção é efectuada periodicamente. Se uma projecção se mantiver incorrecta durante um determinado período de tempo, então uma mudança de *grupo pesado* é necessária. Para determinar o *grupo pesado* de destino, todos os membros do *grupo ligeiro* escolhem um candidato adequado, de entre os *grupos pesados* aos quais o seu processo pertence, e enviam a sua escolha para o coordenador do *grupo ligeiro*, que escolhe o melhor. Se nenhum existir então o coordenador decide pela criação de um *grupo pesado* novo com uma vista idêntica à do *grupo ligeiro*. Depois de decidido o *grupo pesado* de destino é pedida uma mudança de *grupo pesado* e a partir deste ponto o modulo de *Lwg_switch* passa a funcionar.

Estes são os módulos mais importantes, existindo mais alguns auxiliares. Destes o mais importante, embora o menos complexo, é o responsável pelo colocação dos

cabeçalhos do serviço de Grupos Ligeiros nas mensagens que são trocadas entre os membros. A importância deste modulo deve-se à influencia que o manuseamento das mensagens tem no funcionamento do reciclador automático de memória e desse modo no desempenho.

Finalmente temos o *Serviço de Nomes*. Este usa um protocolo muito simples, usando um grupo do Ensemble para manter a coerência entre as réplicas. O protocolo de comunicação com o serviço de Grupos Ligeiros é igualmente muito simples. A comunicação do serviço de Grupos Ligeiros com o Serviço de Nomes não usa nenhum mecanismo de fiabilidade, aumentando unicamente a probabilidade de recepção das mensagens pelo envio para todas as réplicas conhecidas. Já a comunicação no sentido inverso usa um mecanismo simples de retransmissão, oferecendo um certo grau de fiabilidade. Para diminuir o número de mensagens agrupa-se todas as notificações para o mesmo *grupo pesado* numa única mensagem.

As réplicas do Serviço de Nomes funcionam nos mesmos processos dos servidores de "*gossip*" (Hayden & Rodeh, 2000) do Ensemble. Estes últimos são usados pelos *grupos pesados* para detectarem vistas concorrentes. A este servidor com capacidade extendida chamou-se de "egossip".

A concretização do Serviço de Nomes foi desenvolvida tendo como principal prioridade a simplicidade e não o desempenho, pelo que poderia ser melhorada.

4.2.5 Algoritmos

O serviço de Grupos Ligeiros executa três algoritmos principais que serão descritos de seguida.

O primeiro algoritmo é o que permite reunificar vistas concorrentes de um *grupo ligeiro* projectadas no mesmo *grupo pesado*. O algoritmo engloba também a detecção destas vistas concorrentes. O algoritmo compreende cinco passos:

1. Sempre que é entregue uma nova vista do *grupo pesado*, cada coordenador dos *grupos ligeiros* projectados envia uma mensagem para o coordenador do *grupo pesado* com a vista de *grupo ligeiro* entregue.

2. O coordenador do *grupo pesado* guarda todas as vistas de *grupo ligeiro* recebidas.

3. Periodicamente, o coordenador do *grupo pesado* analisa as vistas de *grupos ligeiros* guardadas à procura de vistas concorrentes de um *grupo ligeiro*. Se existirem, envia uma mensagem com as vistas que precisam de se reunificar. A mensagem contém igualmente as vistas reunificadas. A seguir, o coordenador pede uma mudança de vista do *grupo pesado*, ao Ensemble.

4. Após receber a mensagem com as vistas reunificadas, cada membro de um *grupo ligeiro* em causa guarda a vista reunificada, mas continua a operar normalmente. Se o *grupo ligeiro* encontrava-se a meio do processo de mudança de *grupo pesado* então o processo é interrompido.

5. Quando uma nova vista do *grupo pesado* é entregue, os membros do *grupo ligeiro* entregam a nova vista reunificada.

O segundo, e algo mais complicado, é o de mudança de *grupo pesado*. Quando um membro de um *grupo ligeiro* decide que é necessário mudar do *grupo pesado* corrente para um *grupo pesado* destino o algoritmo é iniciado:

1. O membro que decide efectuar a mudança, envia uma mensagem de *begin switch* para todos os membros do *grupo ligeiro*.

2. Após o envio, ou a recepção, da mensagem de *begin switch*, é pedido ao módulo *Hwg* que se junte, ou crie, o *grupo pesado* de destino. Uma cópia do *grupo ligeiro* é então projectada no novo *grupo pesado*. Esta cópia ainda não permite o funcionamento normal.

3. Sempre que ocorre uma mudança de vista no *grupo pesado* de destino é verificado se todos os membros do *grupo ligeiro* já se encontram no *grupo pesado*. Quando isso acontecer é enviado no *grupo pesado* corrente uma mensagem de *Ok*.

4. Quando o coordenador recebe uma mensagem de *Ok* de todos os membros do *grupo ligeiro*, pede uma mudança de vista do *grupo pesado* original.

5. Após receber mensagens de *Ok* de todos os outros membros, quando é recebida uma nova vista do *grupo pesado* original o *grupo ligeiro* deixa este, eliminando a

projecção existente. Neste ponto termina a operação no *grupo pesado* original.

6. Após o fim da operação no *grupo pesado* original, o coordenador do *grupo ligeiro* envia uma mensagem de *end switch* no *grupo pesado* de destino e pede uma mudança de vista deste.

7. Após a recepção da mensagem de *end switch* e da posterior entrega de uma nova vista do *grupo pesado* de destino, é entregue a vista do *grupo ligeiro* já no *grupo pesado* de destino e a operação do grupo retoma a normalidade.

Existem mais alguns pormenores importantes. O primeiro é que, como já foi referido anteriormente, se for recebida uma mensagem para reunificar vistas concorrentes do *grupo ligeiro* no *grupo pesado* corrente o algoritmo é abortado, para evitar um estado incoerente. O segundo é que até ao fim da operação no *grupo pesado* corrente não existe nenhuma restrição ao funcionamento normal do *grupo ligeiro*, podendo ser enviadas e recebidas mensagens e até efectuadas mudanças de vistas do *grupo ligeiro*, desde que não sejam motivadas por reunificação de vistas concorrentes.

Finalmente, temos o algoritmo que reunifica vistas concorrentes de um *grupo ligeiro* projectadas em *grupos pesados* diferentes. A detecção destas vistas concorrentes é feita pelo Servidor de Nomes, que depois notifica o serviço de Grupos Ligeiros. É composto por quatro passos:

1. Quando o coordenador do *grupo pesado* recebe a notificação do Servidor de Nomes da existência de vistas concorrentes, envia uma mensagem de *remote merge* com o identificador do *grupo ligeiro* e dos *grupos pesados* onde as vistas se encontram.

2. Quando o coordenador do *grupo ligeiro* recebe a mensagem do coordenador do *grupo pesado*, compara os dois identificadores de *grupo pesado* e decide de forma determinística qual o *grupo pesado* onde o *grupo ligeiro* vai ser projectado.

3. É iniciada uma mudança de *grupo pesado*.

4. Após a mudança de *grupo pesado*, o processo de reunificação termina com a execução do algoritmo de reunificação de vistas concorrentes no mesmo *grupo pesado*, descrito anteriormente.

De notar que se o processo demorar demasiado tempo, o Servidor de Nomes pode reenviar a notificação, sendo da responsabilidade do coordenador do *grupo ligeiro* ignorar estas notificações repetidas.

4.2.6 Análise resumida

A concretização dos Grupos Ligeiros no Ensemble sofreu em grande medida da limitação das composições de protocolos serem estritamente verticais. Isto impossibilita a existência de camadas que partilhadas entre pilhas e dessa forma que o serviço de Grupos Ligeiros seja concretizado como mais um protocolo presente na composição. Dado esta impossibilidade, a única solução foi concretizar o serviço como um novo interface de comunicação com a aplicação. Este, por razões de transparência, oferece para as aplicações um interface igual ao original. Esta solução sofre de diversos inconvenientes, nomeadamente: a impossibilidade de aceder a certos eventos, que circulam na pilha, que ajudariam ao correcto funcionamento do serviço; e a deficiente flexibilidade de composição, visto que a composição de protocolos usada pelos *grupos ligeiros* tem que ser igual à composição usada pelos *grupos pesados* onde estão projectados.

4.3 Grupos Ligeiros no Appia

A concretização dos Grupos Ligeiros no Appia culmina a aplicação do serviço a diferentes plataformas .

4.3.1 Posicionamento

A possibilidade de uma camada no Appia pertencer a vários canais, ou pilhas de comunicação, possibilita que o serviço de Grupos Ligeiros seja concretizado como uma sessão que é partilhada por todos os canais, ou grupos, que pretendam usar esse serviço. Na concretização efectuada, o posicionamento desta sessão num canal não é fixo, embora esteja limitado por algumas restrições. Uma delas é, como seria de esperar

visto extender os mecanismos de Comunicação em Grupo, ter que estar por cima das camadas que fornecem esse serviço. O posicionamento aconselhado é imediatamente por cima dessas, ou seja, por cima da camada *VSync*.

O posicionamento utilizado têm repercussões importantes sobre as características dos *grupos pesados*, porque estas são definidas pelas camadas que se encontram por baixo da camada do serviço de Grupos Ligeiros. Na concretização no Appia, um *grupo pesado* é um semi-canal que começa no serviço de Grupos Ligeiros até ao fundo, enquanto um *grupo ligeiro* é um semi-canal que começa no topo até ao serviço de Grupos Ligeiros. O serviço torna-se assim um encaminhador entre os semi-canais dos *grupos ligeiros* e dos *grupos pesados* (Figura 4.3). Isto permite que cada *grupo ligeiro* projectado num *grupo pesado* tenha características diferentes, por exemplo, em dois *grupos ligeiros* projectados no mesmo *grupo pesado*, um pode ter ordenação total das mensagens enquanto outro pode ter apenas ordenação FIFO. Esta flexibilidade extra não advém, nem influencia, a concretização do serviço de Grupos Ligeiros e advém do próprio modelo.

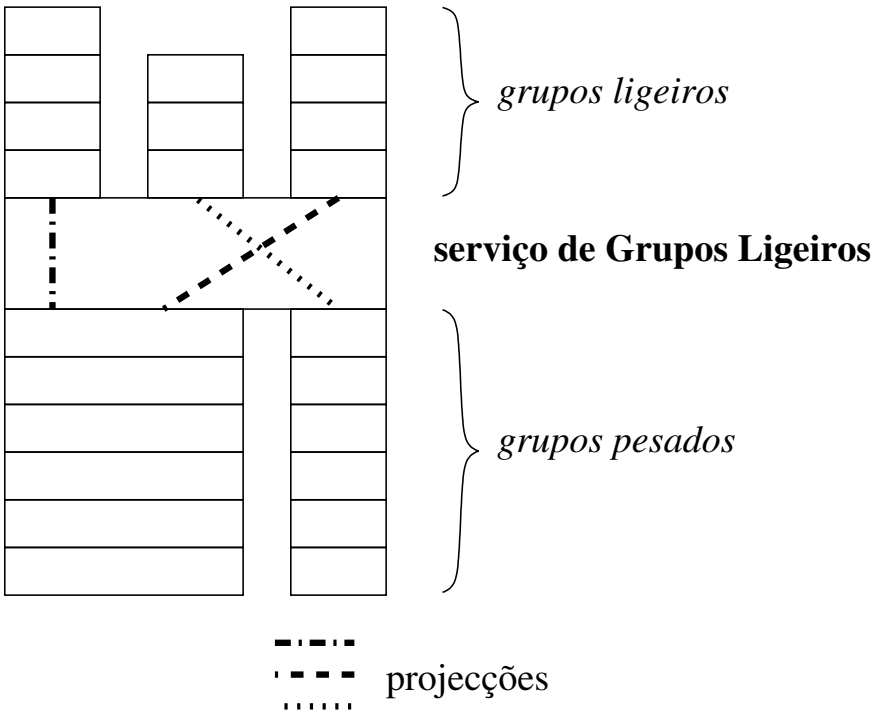

Figura 4.3: Exemplo de projecções entre *grupos ligeiros* e *grupos pesados*.

O encaminhamento é estabelecido por vista, ou seja, durante a utilização de uma determinada vista do *grupo ligeiro* o encaminhamento é constante. Isto permite que os

eventos sejam encaminhados de forma eficiente, pois não é necessário efectuar nenhum processamento adicional. No entanto, os Grupos Ligeiros são um serviço dinâmico, em que as projecções entre *grupos ligeiros* e *grupos pesados* vão sendo corrigidas de forma a manter o sistema eficiente. Isto implica que periodicamente as projecção sejam avaliadas. Quando se determina uma projecção já não é adequada, é iniciado o processo de mudança de projecção, que é executado no âmbito de uma mudança de vista. Assim, é possível manter de forma relativamente simples as propriedades de *sincronia na vista* que norteiam o sistema.

O Appia foi desenvolvido com a capacidade de entregar uma vista sem que isso implique que a aplicação possa enviar mensagens, ou seja, mantendo o grupo bloqueado. Como foi referido anteriormente, para garantir a *sincronia na vista* durante o processo de mudança de vista é necessário bloquear o grupo, ou seja, impedir que os membros enviem mensagens, para garantir que o processo termina. Normalmente, esse bloqueio termina quando a nova vista é entregue. Mas nos Grupos Ligeiros, como as mudanças de vista dos *grupos ligeiros* podem ser independentes das dos *grupos pesados* nos quais estão projectados, isso pode levar a que seja preciso entregar uma nova vista de um *grupo ligeiro* mantendo o bloqueio, porque o *grupo pesado* está bloqueado. Esta entrega de uma nova vista sem desbloquear o grupo é feita no Appia pelo evento BLOCKEDVIEW.

Isto permitiu que fosse concretizado o mecanismo de *esvaziamento por software* ("software-flush") descrito no conceito original (Guo & Rodrigues, 1997). O esvaziamento por software, funciona da seguinte forma:

1. Quando é necessária uma nova vista do *grupo ligeiro*, o coordenador envia para todo o grupo uma mensagem a indicar que se vai mudar a vista. A mensagem pode conter a nova vista.

2. Como para garantir a *sincronia na vista* é necessário bloquear o envio de mensagens, cada membro após receber esta mensagem bloqueia-se e envia a indicação que está bloqueado para todos os outros membros do grupo.

3. Quando cada membro tiver conhecimento que todos os membros do grupo estão bloqueados, entrega a nova vista.

Para que este mecanismo funcione correctamente é necessário que o *grupo pesado* ofereça *sincronia na vista* e que a comunicação entre cada dois membros do grupo respeite a ordenação FIFO. O esvaziamento por software possibilita que seja efectuada uma mudança de vista do *grupo ligeiro* sem obrigar a uma mudança de vista do *grupo pesado*, mantendo as regras da *sincronia na vista*. Isto oferece uma optimização do sistema, pela redução: no número de vezes que esse processo relativamente pesado é executado; e da interferência entre os vários *grupos ligeiros* projectados no mesmo *grupo pesado*.

4.3.2 Estrutura

A concretização dos Grupos Ligeiros no Appia foi dividida num conjunto de módulos/classe, apresentados na Figura 4.4.

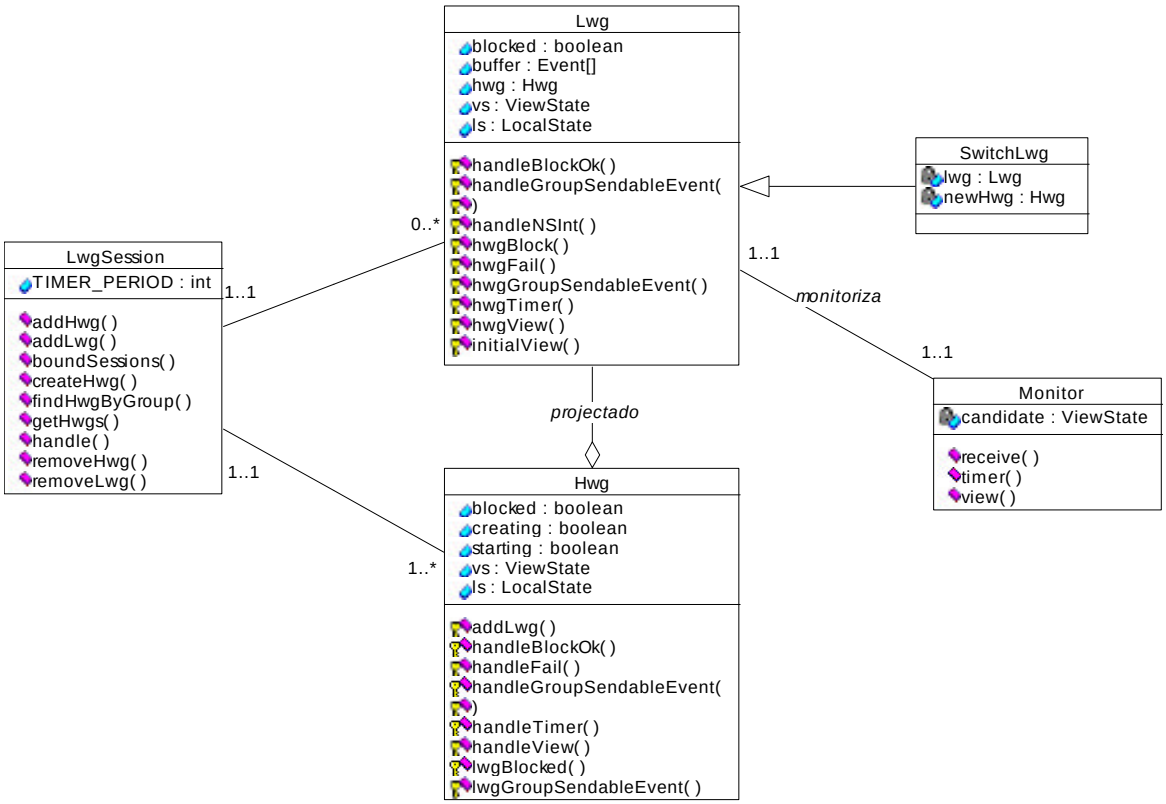

Figura 4.4: Estrutura da concretização dos Grupos Ligeiros no Appia.

O serviço de Grupos Ligeiros foi concretizado como uma camada do Appia. A correspondente sessão é partilhada pelos diversos canais que constituem os *grupos ligeiros*

e *grupos pesados*. Como se trata de uma sessão Appia, toda a interacção é feita através de eventos. O ponto de recepção de eventos é o método handle. Neste, os eventos são analisados e consoante o seu tipo e a sua direcção são entregues ao objecto responsável pelo *grupo pesado* correspondente, da classe HWG, ou ao objecto responsável pelo *grupo ligeiro* correspondente, da classe LWG. Por exemplo, os eventos da classe *GroupSendableEvent* com direcção descendente são entregues ao objecto da classe LWG correspondente, enquanto os ascendentes são entregues ao objecto da classe HWG correspondente. Os *grupos pesados* e *grupos ligeiros* são seleccionados com base no canal do evento, sendo mantido uma projecção entre canais e *grupos ligeiros* ou *grupos pesados*.

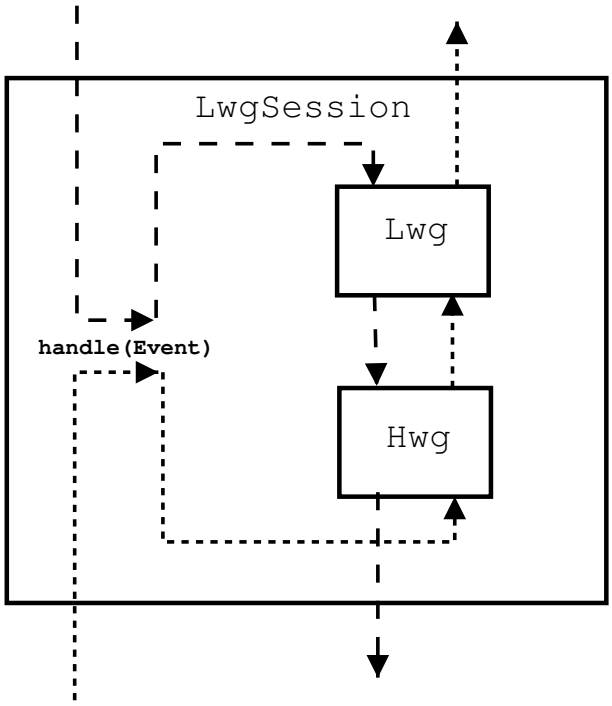

Figura 4.5: Fluxo percorrido pelos eventos no serviço de Grupos Ligeiros concretizado no Appia.

Após um evento ser entregue pelo handle ao *grupo pesado*, é retirada a informação sobre a que *grupo ligeiro* pertence e entregue a este. Após o *grupo ligeiro* o processar, é colocado no canal correspondente ao *grupo ligeiro*, ao qual por definição a sessão do serviço de Grupos Ligeiros pertence, e enviado para cima. Por outro lado, quando o handle entrega um evento a um *grupo ligeiro*, é colocado no evento a identificação do *grupo ligeiro*, e entregue o evento ao *grupo pesado* no qual está projectado. O evento é então colocado no canal correspondente ao *grupo pesado* e enviado para baixo. De notar

que em certas circunstâncias o canal do *grupo ligeiro* pode ser o mesmo que o canal do *grupo pesado*, pelo que não é necessário efectuar a troca de canal. A Figura 4.5 apresenta um esquema do fluxo percorrido pelos eventos.

Cada *grupo pesado* mantém informação sobre a sua vista e sobre quais os *grupos ligeiros* que estão projectados. Cada *grupo ligeiro* por seu turno mantém informação sobre a sua vista e sobre o *grupo pesado* no qual está projectado.

4.3.3 Funcionamento

4.3.3.1 Início

Quando a aplicação pretende se juntar a um grupo, cria um novo canal com as camadas que oferecem o suporte à Comunicação em Grupo e com as propriedades que pretende. Se estiver a usar os Grupos Ligeiros terá que ter uma sessão do serviço de Grupos Ligeiros no canal. Esta terá que ser colocada explicitamente no mesmo. De salientar que poderão existir várias instâncias do serviço de Grupos Ligeiros a funcionar simultaneamente e independentemente. O grupo da aplicação será um *grupo ligeiro* e o canal criado será o canal desse *grupo ligeiro*. Quando o canal é inicializado, as sessões que já estão no canal podem colocar as outras sessões, desde que respeitem as camadas correspondentes. O serviço de Grupos Ligeiros preenche as camadas abaixo dele com as camadas de um *grupo pesado* cujo canal seja compatível. Se existir mais de um *grupo pesado* que satisfaça essa condição, é escolhido aquele que tiver a vista com mais membros, de forma a aumentar a probabilidade dos membros do *grupo ligeiro* pertencerem ao *grupo pesado*. De notar que isto é o máximo que se pode fazer nesta altura pois o serviço de Grupos Ligeiros ainda não sabe a filiação do *grupo ligeiro*, que só será conhecida quando o grupo for inicializado.

Depois do canal estar inicializado, a aplicação irá iniciar o grupo, enviando um evento do tipo GROUPINIT. Quando este evento chega ao serviço de Grupos Ligeiros, este poderá finalmente escolher o *grupo pesado* onde irá projectar o *grupo ligeiro*. Se não existir um *grupo pesado* compatível, tanto em termos de canal como de vistas, terá que ser criado um novo. Este novo *grupo pesado* terá as mesmas camadas que se encon-

tram abaixo do serviço de Grupos Ligeiros no canal do *grupo ligeiro*, só que com novas sessões, e terá como topo a camada do serviço de Grupos Ligeiros.

Quer seja um *grupo pesado* já existente ou um novo, a nova projecção é guardada e a notificação dessa nova projecção é enviada para Serviço de Nomes.

4.3.3.2 Monitorização

Devido ao carácter dinâmico dos grupos, com novos membros a juntarem-se e outros a saírem, para minimizar a interferência entre os *grupos ligeiros* projectados no mesmo *grupo pesado*, as projecções devem ser avaliadas ao longo do tempo. Se se verificar que uma projecção já não satisfaz os requisitos descritos na Secção 2.2.1, então uma nova projecção deve ser criada, sendo se necessário, criado um novo *grupo pesado*.Este novo *grupo pesado* é criado da mesma forma que o criado no início. Os critérios de decisão são iguais aos da concretização no Ensemble descritos na Secção 4.2.

Além das projecções, também a composição da vista do *grupo pesado* deve ser avaliada ao longo do tempo. Isto porque à medida que os *grupos ligeiros* projectados num *grupo pesado* vão mudando ou terminando, poderemos chegar a um ponto em que um dos membros do *grupo pesado* não pertence a nenhum dos *grupos ligeiros* que estão projectados nesse *grupo pesado*. Nesse caso o membro deve sair do *grupo pesado*.

A monitorização das projecções e da composição das vistas dos *grupos pesados* é efectuada periodicamente por *grupo pesado*, ou seja cada *grupo pesado* tem um temporizador periódico que define quando a avaliação de ambos os factores é efectuada.

4.3.3.3 Mudança de *grupo pesado*

Sempre que for necessário mudar um *grupo ligeiro* de um *grupo pesado* para outro, é executado um protocolo especifico para garantir a correcção da mudança, isto é que não sejam violadas as regras da *sincronia na vista*. Para garantir a correcção, e a transparência, o processo de mudança de *grupo pesado* decorre no âmbito de uma mudança de vista do *grupo ligeiro*. O coordenador da vista do *grupo ligeiro*, o membro com o índice mais baixo, envia uma notificação aos outros membros a indicar que se

vai efectuar a mudança para o *grupo pesado* de destino e a seguir inicia um processo de *esvaziamento por software*. Todos os membros do *grupo ligeiro* que ainda não são membros de *grupo pesado*, juntam-se a este. Quando o *grupo ligeiro* estiver bloqueado e tiver sido recebida uma nova vista do *grupo pesado* com todos os membros do *grupo ligeiro*, o grupo é desbloqueado e uma nova vista, já no novo *grupo pesado*, é entregue.

A decisão sobre qual vai ser o *grupo pesado* de destino é tomada de duas formas. Se o objectivo era unir duas vistas do *grupo ligeiro* que se encontravam em *grupos pesados* diferentes, então escolhe-se o *grupo pesado* da vista mais antiga. O coordenador da vista mais recente inicia a mudança, indicando o *grupo pesado* da outra vista como o destino. Se por outro lado a intenção é mudar para um *grupo pesado* mais adequado, então cada membro do *grupo ligeiro* avalia os *grupos pesados* a que pertence e escolhe o mais adequado, se existir um. A escolha de cada membro é enviada para o coordenador do *grupo ligeiro*, que de entre os vários candidatos escolhe o mais indicado e inicia a mudança para esse *grupo pesado*.

4.3.3.4 União de vistas concorrentes de um *grupo ligeiro* no mesmo *grupo pesado*

Este processo é simples e inicia-se quando é entregue uma nova vista do *grupo pesado*. Todos os membros do *grupo pesado* propagam as vistas dos *grupos ligeiros* que têm projectados. Quando cada membro tiver recebido essa informação de todos os outros membros, pode determinar quantas vistas existem de cada *grupo ligeiro* projectado. Se existir mais do que uma, um processo de união é iniciado. Este começa por bloquear os membros das vistas em causa, de forma igual à usada no processo de esvaziamento por software. A diferença em relação a este processo reside no facto de que cada membro espera por todos os membros que pertencem a qualquer uma das vistas que irão seu unidas. Quando todos tiverem confirmado, é entregue a nova vista resultante da união. De salientar que cada membro determina a mesma composição da nova vista, porque a nova vista é a concatenação das vistas a unir, assumindo que estas estão ordenadas pelo índice do coordenador das mesmas na vista do *grupo pesado*.

4.3.4 Serviço de Nomes

O Serviço de Nomes concretizado não assenta num servidor, ou conjunto de servidores replicados, dedicados a esse propósito. Em vez disso, utilizou-se a capacidade do servidor de "gossip" já existente para efectuar uma difusão global, como forma de propagar a informação que o servidor de nomes iria conter. Assim em cada pilha de um *grupo pesado* existe uma camada, chamada de *NS*, que envia através do mecanismo de "gossip" a identificação dos *grupos ligeiros* que estão projectados nesse *grupo pesado*, bem como a identificação do próprio *grupo pesado*. Esta informação irá chegar a todas as camadas *NS* existentes, que poderão detectar quando existem vistas concorrentes dos *grupos ligeiros* projectados no seu *grupo pesado*, notificando o *grupo ligeiro* desse facto. A informação é difundida periodicamente e quando existem modificações.

O Serviço de Nomes é assim concretizado como um servidor distribuído pelos vários *grupos pesados*, ficando com a tolerância a partições que o serviço de "gossip" já oferecia. Tal como na concretização no Ensemble, para o funcionamento do serviço de Grupos Ligeiros é sempre necessário o funcionamento do serviço de "gossip", mesmo quando este não seria necessário devido ao meio de comunicação utilizado suportar difusão selectiva.

4.3.5 Análise resumida

A concretização dos Grupos Ligeiros no Appia seguiu na integra o especificado no conceito. A capacidade do Appia de suportar composições complexas, com partilha de sessões entre canais, permitiu que o serviço fosse colocado exactamente como indicado na sua especificação. O serviço de Grupos Ligeiros é uma camada, um protocolo, que é partilhado por várias composições. Um *grupo ligeiro* corresponde à parte de cima da composição, até à camada correspondente ao serviço. Um *grupo pesado* corresponde à parte de baixo da composição, desde a camada correspondente ao serviço. Esta configuração oferece uma grande flexibilidade porque os *grupos ligeiros* já não têm que ter todos a mesma composição que o *grupo pesado* onde estão projectados. Cada *grupo ligeiro* pode ter uma parte da composição diferente, correspondente às suas proprieda-

des específicas, e partilham a composição correspondente ao *grupo pesado*, que são os recursos que os *grupos ligeiros* projectados partilham. Isto simplifica a concretização.

4.4 Sumário

Neste capítulo são apresentadas as concretizações dos Grupos Ligeiros efectuadas nas plataformas Horus, Ensemble e Appia, sendo que as duas últimas são apresentadas em pormenor porque foram concretizadas no âmbito deste trabalho. Para cada concretização é indicada a sua estrutura, funcionamento e o que foi necessário modificar em relação ao conceito original.

5

Discussão e Avaliação

Qualquer avaliação da concretização de um protocolo sobre uma plataforma pode ser feita sob dois prismas: um qualitativo, em que se avaliam as vantagens e desvantagens conceptuais que influenciaram a forma como o protocolo foi concretizado e que auxiliaram, ou impediaram, que essa concretização fosse simples e elegante; e sob um prisma quantitativo, em que se avalia se a plataforma com o protocolo apresenta um resultado em termos de desempenho vantajoso. Neste capítulo serão apresentadas primeiro as avaliações qualitativas das duas plataformas usadas, o Ensemble e o Appia, e depois serão apresentados os resultados da avaliação quantitativa das duas concretizações.

O capítulo termina com uma análise de avaliações efectuadas por terceiros, que incidiam sobre a plataforma Appia.

5.1 Vantagens e desvantagens funcionais

O Ensemble revelou-se um sistema robusto e com um vasto conjunto de protocolos prontos a utilizar. A sua principal vantagem é, no entanto, a sua eficiência com resultados de desempenho muito bons, apesar de utilizar um reciclador automático de memória. Este revelou-se uma faca de dois gumes. Por um lado significou uma velocidade de desenvolvimento inicial relativamente alta, por outro, é necessário um enorme controlo e cuidado por parte do programador, para não deixar que o seu funcionamento afecte de forma muito negativa o desempenho obtido. Estes cuidados têm duas vertentes: usar só as primitivas iterativas da linguagem usada e utilizar correctamente as ferramentas de gestão dos tampões correspondentes às mensagens.

No entanto, a grande desvantagem do Ensemble é o facto de suportar unicamente composições de protocolos estritamente verticais, ou seja, em pilha sem possibilidade de partilha de camadas entre pilhas. Esta limitação é justificada pelas ferramentas de validação formal e optimização da execução, que só suportam essa composição. Embora estas ferramentas tenham bastante valor, as restrições ao nível da composição de protocolos não deixam de ser uma limitação que deveria ser abordada. Esta limitação obrigou a que a concretização efectuada do serviço de Grupos Ligeiros no Ensemble não fosse a melhor em termos do seu posicionamento no sistema. Isto por seu turno limitou as capacidades ao dispor do mesmo, sem acesso a vários eventos só disponíveis às camadas dentro da pilha.

O posicionamento provocou igualmente a perda da flexibilidade na composição dos vários *grupos ligeiros* projectados num *grupo pesado*. Na concretização efectuada, todos os *grupos ligeiros* têm que partilhar exactamente a mesma pilha, ou seja, com as mesmas propriedades que o *grupo pesado*. É impossível ter *grupos ligeiros* com diferentes propriedades projectados no mesmo *grupo pesado*, reduzindo as possibilidades de encontrar a projecção mais correcta entre *grupos ligeiros* e *grupos pesados*.

Embora na concretização efectuada esse problema não se tenha revelado, a experiência com o serviço de Grupos Ligeiros no Appia revelou que a capacidade de adicionar, de forma simples e dinâmica, novos eventos é muito útil. Só que no Ensemble os eventos disponíveis são fixos (cerca de 40) e adicionar um novo implica recompilar todo o sistema. Mesmo depois disso, podem existir problemas devido ao complexo processamento que estes sofrem por parte das diversas camadas.

Um outro problema encontrado prende-se com a incapacidade de entregar uma nova vista mantendo a impossibilidade de enviar mensagens, ou seja, mantendo o grupo bloqueado. Esta limitação prende-se mais com a utilização normal da *sincronia na vista*, que assume que quando uma vista é entregue o grupo fica desbloqueado. No entanto, esta limitação do Ensemble não permitiu a utilização do protocolo de *esvaziamento por software*. Este permitia que fosse correctamente instalada uma nova vista do *grupo ligeiro* sem obrigar a uma mudança de vista do *grupo pesado*. Como isto não é possível (ver Secção 4.2), sempre que é necessário mudar uma vista de um *grupo ligeiro*

é preciso efectuar uma mudança de vista do *grupo pesado*, para garantir a correcção da mudança, o que acarreta um enorme custo em termos de interferência entre os *grupos ligeiros* projectados, que como o *grupo pesado* mudou de vista também têm que mudar de vista. A mudança de vista de um *grupo ligeiro* obriga todos os outros a mudar de vista também.

Outro aspecto é a incapacidade do interface da aplicação do Ensemble de forçar a instalação de uma nova vista com um conjunto de novos membros adicionados, o que origina que quando um *grupo ligeiro* muda de *grupo pesado*, potencialmente cada membro do *grupo ligeiro* terá que se juntar ao novo *grupo pesado* separadamente, o que provocará um número de mudanças de vista igual ao número de membros do *grupo ligeiro*. Isto representa um alto nível de interferência. Embora o Ensemble tente agrupar a junção destes novos membros numa única mudança de vista, isso não é imposto, pelo que na prática essas várias mudanças de vista do *grupo pesado* acontecem.

Já o Appia correspondeu genericamente ao esperado, excepto num grande factor, o desempenho. Este é algo deficiente, em parte devido à linguagem usada na sua concretização, o JAVA. Mas esta não justifica tudo e outras avaliações (apresentadas na Secção 5.3) revelaram que existe igualmente necessidade de maior estudo da concretização da plataforma e seus protocolos.

De resto, o Appia revelou alguns dos seus pontos fortes, como a fácil habituação e desenvolvimento de novos protocolos, já demonstrados na sequência de anteriores usos da ferramenta no ensino do desenvolvimento de protocolos para sistemas distribuídos.

O posicionamento do serviço concretizado no Appia é o ideal, funcionando como um encaminhador entre os "semi-canais" dos *grupos ligeiros* e dos *grupos pesados*. Desta forma o serviço tem acesso a todos os eventos das camadas de suporte à Comunicação em Grupo, permitindo o acesso a mecanismos inacessíveis à concretização no Ensemble, garantindo assim uma maior capacidade e transparência de operação. Com o posicionamento utilizado ganha-se igualmente uma grande flexibilidade na composição dos *grupos ligeiros*, permitindo que *grupos ligeiros* com propriedades diferentes mas filiação semelhantes possam ser projectados no mesmo *grupo pesado*. A diferença reside

unicamente no "semi-canal" do *grupo ligeiro*. A flexibilidade extra advém da própria plataforma e não requer quaisquer mecanismos específicos por parte do serviço de Grupos Ligeiros.

Um aspecto a considerar em futuros melhoramentos do Appia seria a capacidade de mudar a constituição do canal, em termos das sessões que o constituem, em tempo de execução. Esta capacidade permitiria que os "semi-canais" dos *grupos ligeiros* fossem substituídos por canais completos, em que sempre que o *grupo ligeiro* mudava de *grupo pesado* eram mudadas as sessões correspondentes a este. Com isto, deixava de ser necessário efectuar o encaminhamento no serviço de Grupos Ligeiros, poupando-se a mudança de estado que a mudança de canal implica. O serviço de Grupos Ligeiros assumiria um papel meramente monitorizador e de gestão dos diversos canais. No Ensemble já existe a capacidade de mudar as camadas em tempo de execução, mas devido ao posicionamento que o serviço de Grupos Ligeiros teve que tomar não foi possível tirar partido dessa capacidade.

5.2 Análise de desempenho

A análise de desempenho procurou demonstrar as vantagens de utilização do serviço de Grupos Ligeiros, em ambientes com vários grupos de filiação semelhante, sem com isso penalizar significativamente o desempenho noutros ambientes.

Nos testes efectuados utilizou-se a aplicação *Perf*. Esta aplicação existe tanto no Ensemble como no Appia, sendo que a concretização neste último foi baseada na do Ensemble. A aplicação oferece um conjunto de testes, dos quais o mais usado foi o "ring". Neste, numa vista com n membros, cada membro envia k mensagens seguidas, de tamanho m, ficando depois à espera de receber $(n-1)*k$ mensagens de todos os outros membros antes de voltar a enviar k mensagens. Este processo repete-se r vezes. Variando o número de mensagens enviadas de seguida e o tamanho das mesmas, obtêm-se aproximações das seguintes medidas:

latência por ronda que indica quanto tempo demora a completar uma ronda, ou seja, enviar uma mensagem para todos os membros e receber uma de cada um deles.

	k	m
latência	1	0
largura de banda	1	*grande*
débito	*grande*	0

Tabela 5.1: Relação entre parâmetros k e m no teste *ring* da aplicação Perf.

débito por membro que indica quantas mensagens cada membro conseguiu enviar por segundo.

largura de banda do grupo que indica quantos bytes foram trocados por segundo entre os membros de um grupo.

A Tabela 5.1, retirada de (Hayden & Rodeh, 2000), apresenta a relação entre o número de mensagens, o tamanho das mesmas e a medida obtida.

Os testes foram efectuados em quatro PCs Pentium II a 350 MHz ligados por um *hub* Ethernet a 100Mb/s.

5.2.1 Grupos Ligeiros e Ensemble

A Figura 5.1 apresenta a latência por ronda com um número sucessivamente maior de grupos a funcionar concorrentemente. Embora o serviço de Grupos Ligeiros apresente piores resultados com poucos grupos, a partir de seis grupos os resultados apresentam uma melhoria razoável.

A Figura 5.2 mostra o débito por membro, por grupo. Aqui os resultados do uso dos Grupos Ligeiros não são expressivos, com resultados muito iguais aos obtidos sem o serviço de Grupos Ligeiros. Isto deve-se ao facto de os Grupos Ligeiros terem um custo fixo por mensagem. Mesmo assim os ganhos apresentados devem-se à diminuição dos recursos utilizados.

A Figura 5.3 mostra a largura de banda utilizada por cada grupo. Como seria de esperar quantos mais existem, menos largura de banda está disponível para cada um deles. Aqui os Grupos Ligeiros demonstram a sua utilidade, ao apresentarem resultados significativamente melhores com um número de grupos maior. Igualmente de

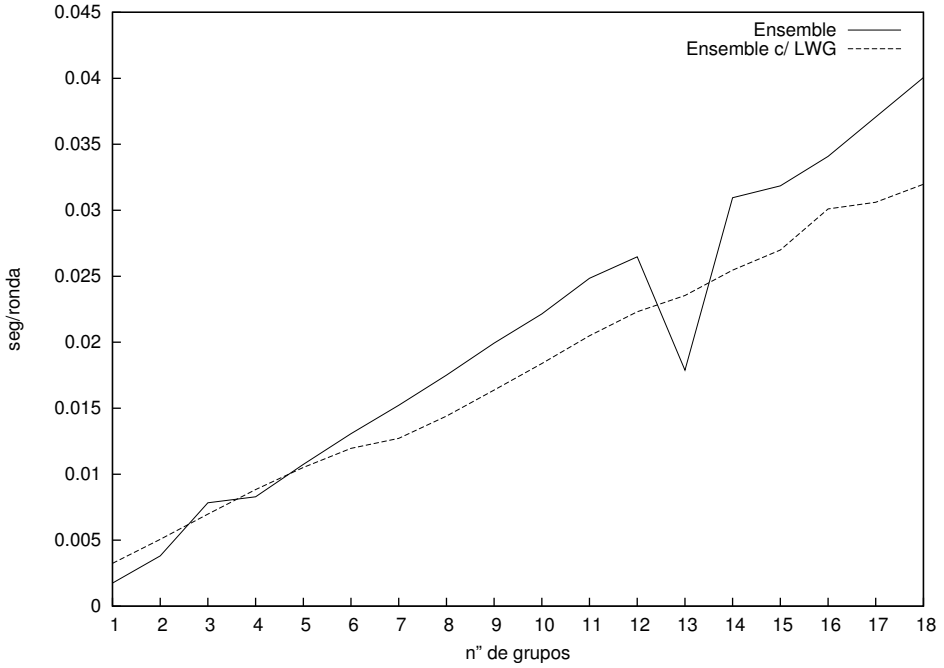

Figura 5.1: Latência por ronda relativamente ao número de grupos a funcionar em paralelo, no Ensemble

realçar o facto de uma melhoria aparecer logo com apenas dois grupos, ao contrário dos testes anteriores em que isso só acontecia a partir dos cinco grupos a operar simultaneamente.

5.2.2 Grupos Ligeiros e Appia

A análise dos resultados obtidos com a concretização dos Grupos Ligeiros no Appia é semelhante à efectuada para o Ensemble.

A Figura 5.4 apresenta a latência por ronda com um número sucessivamente maior de grupos a funcionar concorrentemente. Tal como esperado, com poucos grupos o custo do serviço de Grupos Ligeiros implica que o seu uso induz uma pequena degradação do desempenho. No entanto a partir, de seis grupos começa a revelar uma melhoria significativa. De referir que com mais de quinze grupos não existem resultados para o Appia sem Grupos Ligeiros porque os testes excederam o limite de tempo estipulado para cada teste.

A Figura 5.5 mostra o débito por membro, por grupo. Os resultados são semelhan-

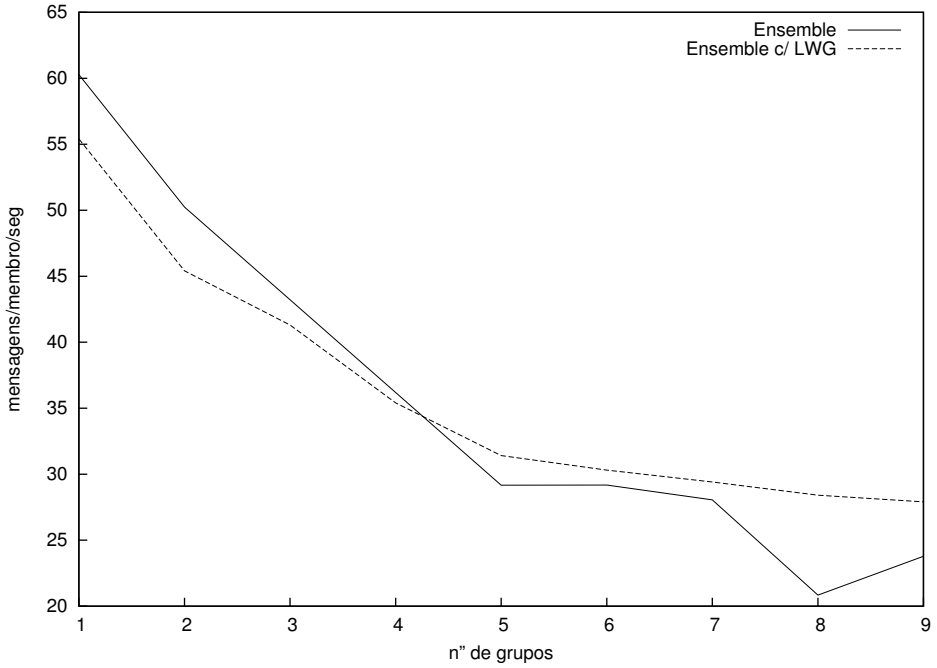

Figura 5.2: Débito por membro relativamente ao número de grupos a funcionar em paralelo, no Ensemble

tes, com o serviço de Grupos Ligeiros a demonstrar novamente as suas vantagens com mais de seis grupos e o Appia sem Grupos Ligeiros a não conseguir terminar o teste no tempo limite, com mais de dezasseis grupos.

A Figura 5.6 mostra a largura de banda utilizada por cada grupo. O importante deste gráfico é demonstrar como o serviço de Grupos Ligeiros ao reduzir o consumo dos recursos de suporte à Comunicação em Grupo permite disponibilizar mais largura de banda para cada grupo. Por essa razão, a melhoria do uso do serviço de Grupos Ligeiros neste cenário é imediata.

5.3 Outras Análises/Avaliações

5.3.1 "Appia vs Cactus"

Uma outra avaliação das plataformas de composição de protocolos é oferecida por (Sergio Mena *et al.* , 2003), em que são comparados os sistemas Appia e Cactus. A escolha destes sistemas deve-se ao facto de serem, segundo os autores, dois sistemas

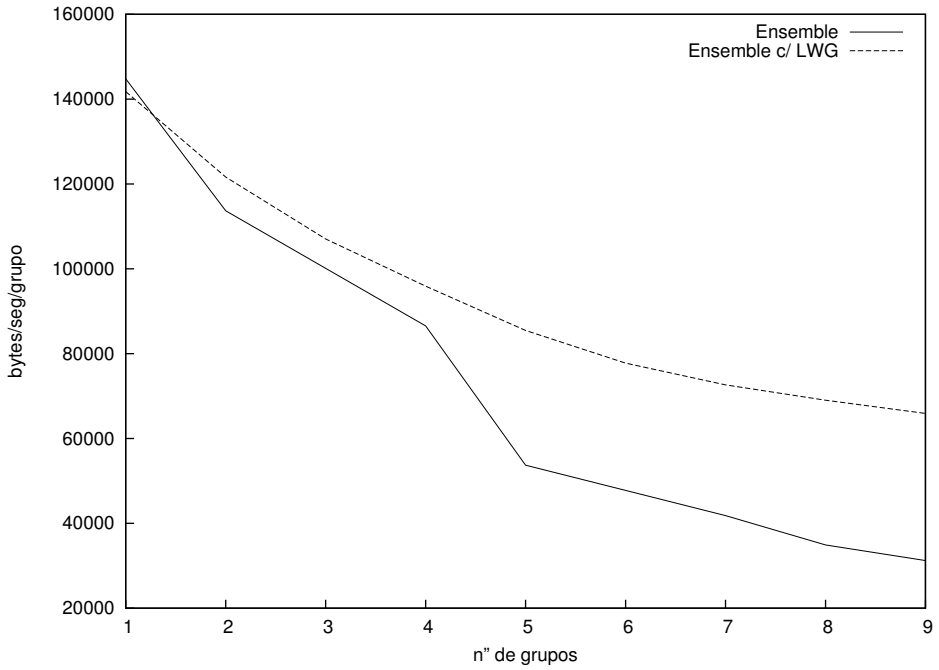

Figura 5.3: Largura de banda usada por cada grupo a funcionar em paralelo, no Ensemble

representativos das duas grande famílias de plataformas de composição, por um lado o Horus, Ensemble, Appia e JavaGroups (JavaGroups, 2004) e por outro x-Kernel e Cactus.

Como forma de comparação, foi concretizado um serviço idêntico de *difusão atómica tolerante a falhas* nas duas plataformas. O serviço foi concretizado a partir de quatro módulos:

Canal Fiável, que concretiza a comunicação fiável entre dois processos e que é baseada no TCP.

Detector de Falhas, baseado na troca de mensagens de "ping".

Consenso, que utiliza os dois módulos anteriores para resolver questões de consenso.

Difusão Atómica, que utiliza os outros módulos para oferecer difusão atómica.

Com o intuito de simplificar a concretização nas duas plataformas e como forma de garantir que em ambas era executado exactamente o mesmo algoritmo, estes módulos foram concretizados cada um numa classe JAVA, independente da plataforma. Dentro

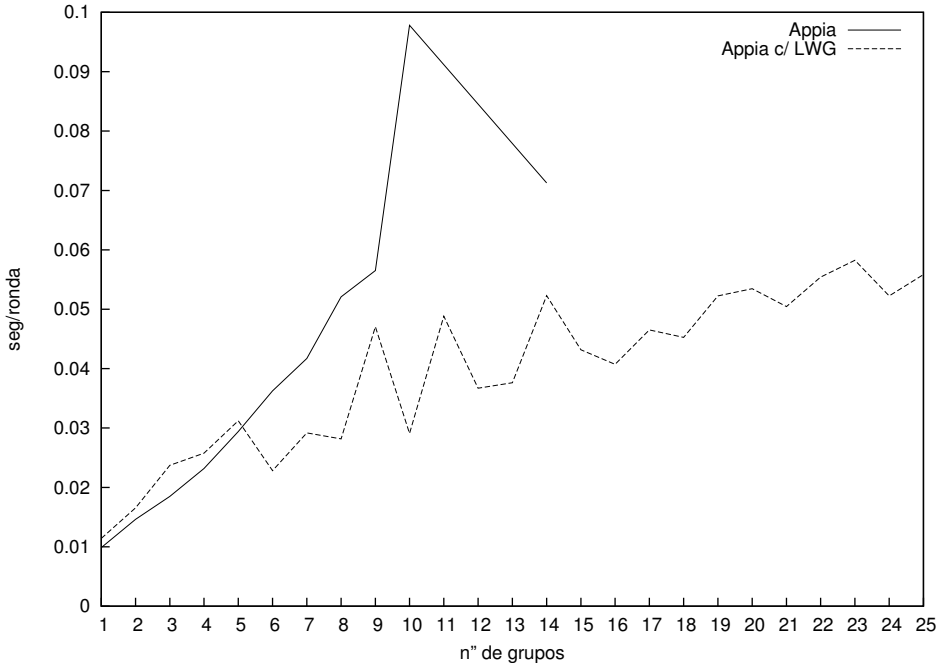

Figura 5.4: Latência por ronda relativamente ao número de grupos a funcionar em paralelo, no Appia

de cada plataforma basta invocar a funcionalidade destas classes. Já a forma como o serviço é colocado dentro da plataforma varia entre as duas concretizações, na medida em que as plataformas também oferecem modelos diferentes. No Cactus o serviço foi concretizado como um único protocolo composto, em que cada um dos módulos referidos é encapsulado dentro de um micro-protocolo. Já no Appia cada módulo foi encapsulado dentro de uma camada, sendo que o serviço corresponde assim a uma pilha com quatro camadas.

Para evitar trocas de contexto entre actividades e para que a comparação seja justa, as concretizações assentam no uso de uma única actividade. No Appia, por definição, esta era a única possibilidade, mas no Cactus teve-se que usar um "semáforo" para controlar o acesso ao protocolo composto e garantir que apenas uma actividade acedia ao mesmo.

O trabalho identifica três características comuns às duas plataformas:

- A estrutura interna dos protocolos, quer sejam sessões do Appia ou micro-protocolos do Cactus, é igual, sendo essencialmente um conjunto de funções para tratamento de eventos e um estado interno, que reagem à ocorrência dos even-

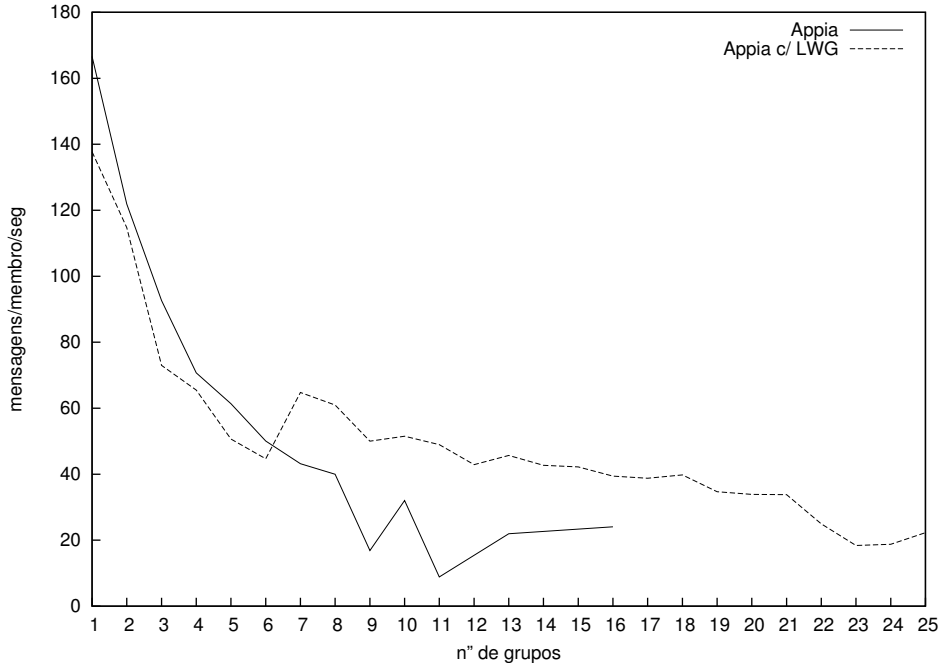

Figura 5.5: Débito por membro relativamente ao número de grupos a funcionar em paralelo, no Appia

tos registados. Foi esta semelhança que permitiu que ambas as concretizações usassem a mesma funcionalidade.

- A informação que é enviada pela rede é encapsulado por uma *mensagem*, onde são colocados e retirados cabeçalhos. Embora a estrutura das mesmas varie o conceito é semelhante.

- Ambos os sistema não oferecem nenhum mecanismo de controlo de fluxo, que regule o ritmo pelo qual os eventos são colocados no sistema.

São igualmente identificadas quatro áreas em que os dois sistemas divergem:

- O Appia usa um modelo de composição hierárquico, em que as sessões podem ser partilhadas por vários canais. A comunicação entre as sessões de um canal tem que seguir a ordem pela qual estas foram organizadas no mesmo, não podendo comunicar directamente com as sessões que não lhe estão adjacentes, mesmo que os eventos possam saltar algumas devido à optimização nos seus caminhos. Já o Cactus usa um modelo cooperativo, em que qualquer micro-protocolo pode comunicar directamente com qualquer outro micro-protocolo,

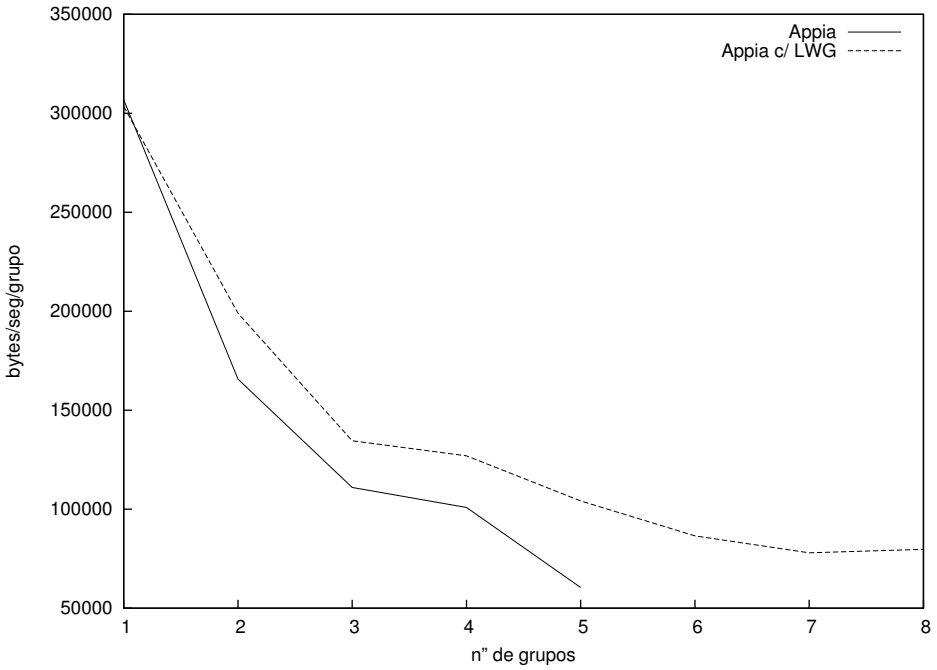

Figura 5.6: Largura de banda usada por cada grupo a funcionar em paralelo, no Appia

dentro do protocolo composto, que deseje.

- No Appia a interacção entre os protocolos só pode ser feita através de eventos, enquanto no Cactus além dos eventos também se pode utilizar informação partilhada, que no entanto não foi utilizada na concretização efectuada. Igualmente diferente é a ordem pela qual os eventos são recebidos pelos protocolos, que no Appia é FIFO e no Cactus não existe, ou seja, no Cactus não existe uma ordem definida pela qual os micro-protocolos recebem os eventos.

- O modelo de concorrência é diferente, com o Appia a suportar apenas uma actividade enquanto o Cactus permite múltiplas actividades, embora a correcta sincronização entre elas tenha que ser feita pelos programadores dos micro-protocolos, sendo que a plataforma não ofereça qualquer ferramenta que auxilie na tarefa.

- O interface com o ambiente é diferente. No Cactus quer a aplicação como a rede têm que adoptar o interface de um protocolo composto, que não é mais que o interface de um protocolo do x-Kernel. Isto limita a aplicação, pois só oferece um mecanismo de envio de mensagens, que não permite fazer distinções entre

dois tipos de mensagens de forma elegante. Por exemplo, considerando uma aplicação que envia mensagens que precisam de ser entregues seguindo uma ordem total, ou mensagens que podem ser entregues por qualquer ordem, o Cactus não oferece mecanismos para distinguir entre as duas sem ser colocando um atributo de tipo na mensagem, que teria que ser conhecido pelos micro-protocolos. No Appia a interacção entre a aplicação e/ou a rede e o ambiente é feita através dos chamados eventos assíncronos. Isto oferece maior flexibilidade, pois o evento que é colocado pode ser de um tipo que exprima as características do mesmo, mas o mecanismo só funciona no sentido da aplicação para o canal, enquanto que a comunicação do canal para a aplicação tem que se feita de uma forma "ad-hoc".

Foram efectuados testes de desempenho das duas concretizações. O primeiro teste efectuado foi de débito relativamente ao tamanho das mensagens enviadas. Foram usados os resultados de usar o TCP directamente como valor de referência. Os resultados indicaram que quer o Appia como o Cactus impõem um custo bastante significativo, sobretudo quando usadas mensagens pequenas. Outro resultado obtido foi que à medida que o tamanho das mensagens aumenta o Cactus melhora mais que o Appia, chegando-se a resultados em que Cactus apresenta um valor quatro vezes melhor. O segundo teste foi de medição do tempo de ida-e-volta de uma mensagem. Foi novamente usado a utilização directa do TCP como referência. Os resultados são análogos aos do primeiro teste, com ambas as plataformas a induzirem um custo significativo, que se nota especialmente nas mensagens de pequena dimensão, e o Cactus a obter resultado melhores que o Appia.

Para compreender os resultados foi usada uma ferramenta de monitorização ("profiling") que indicou que a maior parte do tempo de execução é gasto no tratamento das mensagens, cerca de 84% no Appia e 66% no Cactus, e portanto o acondicionamento da informação na mensagem é o principal condicionador do desempenho de ambas as plataformas. Ambas as plataformas usam os mecanismos de codificação de objectos ("serializing") do JAVA que se revelam algo ineficientes. A Tabela 5.2 apresenta os valores obtidos.

Por deficiência na documentação que acompanha o Appia, os autores deste traba-

Operação	Appia	Cactus
Sincronização de Actividades	7%	24%
Plataforma	4%	8%
Avaliação	5%	2%
Processamento de Mensagens	84%	66%

Tabela 5.2: Resultados da utilização de uma ferramenta de monitorização no Appia e Cactus.

lho não usaram alguns dos mecanismos para codificação existentes na EXTENDEDMESSAGE para resolver exactamente esse problema, o que pode justificar, pelo menos em parte, os resultados obtidos. No entanto, torna-se essencial um estudo mais aprofundado dos mecanismos de acondicionamento usados e suas implicações.

Finalmente são apresentadas guias para o desenvolvimento de futuras plataformas:

- Como muitos eventos destinam-se unicamente de um protocolo para outro, deve haver um mecanismo para que sejam entregues directamente, não usando uma estrutura tão rígida na comunicação entre os protocolos como a hierárquica usada no Appia.

- Qualquer que seja o modelo de concorrência utilizado, deve oferecer as seguintes garantias: sincronização de actividades por omissão, para os programadores dos protocolos não terem que se preocupar com esse aspecto; os eventos devem ser recebidos respeitando uma ordenação FIFO; e transparência do modelo de concorrência, de forma a poder mudá-lo para o que melhor se adaptar as necessidades sem ter de modificar os protocolos.

- O interface entre a aplicação e a plataforma deve seguir aquele usado no Appia, onde a aplicação, ou a rede, podem inserir eventos na plataforma. Já o interface da plataforma para a aplicação deve consistir num conjunto de interfaces comuns alternativos, visto que a especificação de um único interface que se adeqúe a todas as aplicações seria impossível.

Estes três pontos merecem alguns comentários. Em relação ao primeiro à que referir que, ao contrário do que é afirmado, uma análise dos eventos trocados entre cama-

das, durante o funcionamento de um sistema de Comunicação em Grupo, revelou que poucos eventos se destinam unicamente a duas camadas. Além disso, a capacidade de adicionar novos eventos aliada à optimização que é efectuada sobre o caminho percorrido por estes, permitia que se duas camadas quisessem comunicar directamente entre elas o pudessem fazer usando um tipo de evento criado para esse efeito. Por estas razões não é óbvio que essa rigidez na comunicação entre camadas seja uma desvantagem. Já o segundo ponto vem de encontro ao preconizado aquando da concepção do Appia, que levou ao uso de uma única actividade. O problema inerente ao último ponto também já tinha sido detectado durante o desenvolvimento da plataforma Appia. Numa tentativa de resolver parte do problema foi concretizada uma interface para o sistema de Comunicação em Grupo, a COMMAPI.

5.3.2 "Análise de desempenho de plataformas, em JAVA, de suporte à comunicação em grupo"

Outra avaliação de plataformas de composição de protocolos aparece em (Baldoni *et al.* , 2003). Esta centra-se mais nos aspectos de desempenho e justifica-se na crescente procura por soluções que permitam o desenvolvimento de sistemas fiáveis baseados no JAVA. Uma forma de conseguir estes sistemas fiáveis é através da replicação por software. Para garantir a consistência das réplicas usa-se muitas vezes primitivas de Comunicação em Grupo. Assim uma forma de atingir a replicação é usando sistemas de Comunicação em Grupo. A avaliação não distingue a plataforma de composição dos protocolos em si, avaliando o conjunto como um todo.

Os sistemas avaliados são o *Spread* (Amir & Stanton, 1998), Appia e *JavaGroups* (JavaGroups, 2004), enquanto que o Ensemble também é avaliado como referência. O Spread usa uma arquitectura cliente-servidor em que o cliente comunica com um servidor que se encontra na mesma máquina, que participa na comunicação do grupo. O servidor tem um número fixo de protocolos e não permite a configuração ou adição das propriedades pretendidas. O cliente é concretizado em JAVA mas o servidor é concretizado em C. Já o JavaGroups é um sistema totalmente desenvolvido em JAVA e pode ser usado como interface com outro sistema, como o Ensemble, ou utilizar uma

plataforma e protocolos próprios. A plataforma para composição de protocolos do JavaGroups oferece uma composição em pilha.

Para efectuar a avaliação foi concretizado em cada uma das plataformas um sistema de replicação activa em três níveis. Nestes, os clientes e as réplicas não participam na sincronização. Esse trabalho é executado por um nível intermédio, responsável por garantir que todas as réplicas executam exactamente as mesmas operações pela mesma ordem, de forma a que os seus estados se mantenham sincronizados. Para garantir isto, este nível tem que resolver duas questões principais, atomicidade das operações e ordenação global das mesmas. Os sistemas de Comunicação em Grupo já oferecem protocolos que garantem estas propriedades. Na realidade até oferecem vários protocolos que utilizando algoritmos diferentes oferecem as mesmas propriedades. Um exemplo é a ordenação global das mensagens, que é alcançada por um protocolo de ordem total, do qual existem muitas alternativas a escolher consoante as necessidades. Na avaliação, quer o JavaGroups como o Apppia foram usados com duas versões de protocolos de ordem total, uma baseada num sequenciador e outra distribuída (no JavaGroups baseada num "token" e no Appia baseada em duas fases).

Foram efectuados testes que mediram a latência na execução de uma operação no cliente e na obtenção de uma ordenação no serviço de Comunicação em Grupo. Foram executados três conjuntos de testes para avaliar a capacidade de escalar do sistema em termos do número clientes, de réplicas, e de elementos no sistema de Comunicação em Grupo. Foram sempre obtidos resultados para o Ensemble, Spread, Appia com ordenação baseada num sequenciador, Appia com ordenação distribuída, JavaGroups com ordenação baseada num sequenciador e JavaGroups com ordenação distribuída. Os resultados, como esperado, indicam que o Ensemble apresenta o melhor desempenho, mas as plataformas baseadas em JAVA conseguem, em algumas condições, resultados muito perto dos do Spread, que é concretizado parcialmente em C. Já nas plataformas em JAVA, o algoritmo de ordenação total distribuída do Appia revelou-se melhor que o correspondente do JavaGroups, mas ambos apresentam resultados inferiores aos baseados num sequenciador. Com estes o Appia apresenta melhores resultados que o JavaGroups excepto quando se aumenta o número de clientes.

Em termos gerais o Appia apresenta melhores resultados que o JavaGroups, pelo menos em grupos com até oito membros, já que não foram testados grupos maiores.

5.4 Sumário

Neste capítulo foram apresentadas as vantagens e desvantagens que cada uma das plataformas teve na concretização do serviço de Grupos Ligeiros, no fundo o que se teve que mudar do conceito original dos Grupos Ligeiros para permitir a sua concretização nas plataformas. Depois, foi apresentada uma avaliação dos resultados dos testes de desempenho efectuados sobre as duas plataformas. Finalmente, foram apresentados duas outras avaliações que também analisavam o comportamento da plataforma Appia.

6

Conclusão

Esta dissertação descreve a concretização e a avaliação da plataforma de suporte à composição e execução de protocolos *Appia*. Esta plataforma foi originalmente desenhada por terceiros (Miranda, 2001), com o objectivo de facilitar a configuração de sistemas de comunicação com requisitos de coordenação entre canais. Pretendeu-se com o trabalho aqui descrito aferir a adequabilidade dos mecanismos de composição propostos pelo Appia assim como a eficiência destes mecanismos em tempo de execução.

De modo a dar substância à avaliação qualitativa dos mecanismos de composição, optou-se por concretizar um serviço de comunicação em grupo com requisitos de coordenação multi-canal, o serviço de Grupos Ligeiros. Este serviço foi concretizado na plataforma Appia e numa plataforma alternativa, o Ensemble. Para concretizar o serviço de Grupos Ligeiros sobre o Appia foi necessário desenvolver previamente, não só uma concretização da plataforma, mas também uma concretização de um serviço de Comunicação em Grupo fiável para esta plataforma. Refira-se que, cronologicamente, a concretização do serviço de Grupos Ligeiros sobre o Ensemble precedeu a concretização do Appia, o que permitiu aplicar a experiência adquirida com a utilização da primeira plataforma no desenvolvimento da segunda.

Sintetizando, os principais resultados deste trabalho foram:

- O desenvolvimento de uma concretização da plataforma Appia assim como uma composição de protocolos que oferece um serviço de Comunicação em Grupo com *sincronia na vista* para esta plataforma. Este protótipo está disponível na Internet e tem sido extensivamente utilizado em projectos de investigação do grupo

DIALNP[1]. Tem, desde 2001, sido utilizado para ensino na cadeira de Tolerância a Faltas Distribuída da Licenciatura em Informática da FCUL. A plataforma foi também testada e avaliada por alguns grupos de investigação fora de Portugal.

Com base na experiência reportada nesta dissertação e obtida através dos restantes projectos em que o Appia foi utilizado, podemos afirmar que a plataforma é de utilização simples e intuitiva, sendo particularmente útil no suporte ao desenvolvimento rápido de protótipos.

Infelizemente, o desempenho da plataforma e dos protocolos ficou aquém do esperado. Se por um lado, seria previsivel que, devido a utilização da linguagem JAVA, o desempenho do sistema fosse inferior ao desempenho do Ensemble, verificaram-se também limitações quando comparado com outras plataformas desenvolvidas na mesma linguagem.

- O desenvolvimento de duas concretizações do serviço de Grupos Ligeiros, uma sobre o Ensemble e outra sobre o Appia. Este exercício permitiu confirmar que a plataforma Appia possui mecanismos de composição significativamente mais expressivos do que a plataforma Ensemble. Em consequência, a concretização deste serviço no Appia é significativamente mais simples e elegante. Uma limitação detectada no Appia prende-se com a incapacidade de mudar a constituição de um canal, em termos de sessões, durante o seu funcionamento. Essa funcionalidade permitiria reduzir o número de canais utilizados e ainda permitir uma melhor projecção entre os *grupos ligeiros* e o *grupos pesados*.

Apesar de diverso trabalho efectuado aquando da concretização do Appia (gestão explicita de objectos, redução do número de objectos necessários, etc.), no sentido de retirar um bom desempenho da plataforma, isso ainda não foi alcançado plenamente. Assim, existe a necessidade de em trabalho futuro, efectuar um estudo muito mais aprofundado dos actuais pontos de estrangulamento da plataforma, bem como optimizar a concretização dos protocolos mais utilizados, de modo a aproximar, ou mesmo superar, o desempenho de sistemas semelhantes desenvolvidos na mesma linguagem.

[1]Distributed ALgorithms and Network Protocols, um dos grupo de investigação do Laboratório de Sistemas Informáticos de Grande Escala.

Bibliografia

AMIR, Y., & STANTON, J. 1998. *The spread wide area group communication system*. Tech. rept. 98-4. CNDS.

BALDONI, ROBERTO, CIMMINO, STEFANO, MARCHETTI, CARLO, & TERMINI, ALESSANDRO. 2003. Performance Analysis of Java Group Toolkits: A Case Study. *Pages 49–60 of: Scientific Engineering for Distributed Java Applications, International Workshop, FIDJI 2002. Luxembourg-Kirchberg, Luxembourg*. Lecture Notes in Computer Science, vol. 2604. Springer.

BHATTI, NINA T., HILTUNEN, MATTI A., SCHLICHTING, RICHARD D., & CHIU, WANDA. 1998. Coyote: A System for Constructing Fine-Grain Configurable Communication Services. *ACM Transactions on Computer Systems*, **16**(4), 321–366.

BIRMAN, K., & VAN RENESSE, R. (eds). 1994. *Reliable Distributed Computing With the ISIS Toolkit*. IEEE CS Press.

CACTUS. 2004. *The Cactus Project*. http://www.cs.arizona.edu/cactus/.

GARBINATO, B., FELBER, P., & GUERRAOUI, R. 1996. Protocol Classes for Designing Reliable Distributed Environments. *In: European Conference on Object Oriented Programming (ECOOP)*, vol. 1098. Springer Verlag (LNCS).

GUO, K., & RODRIGUES, L. 1997. Dynamic Light-Weight Groups. *Pages 33–42 of: Proceedings of the 17th International Conference on Distributed Computing Systems*. IEEE, Balitmore, Maryland, USA.

HAYDEN, M. 1998. *The Ensemble System*. Ph.D. thesis, Cornell University, Computer Science Department.

HAYDEN, MARK, & RODEH, OHAD. 2000. *Ensemble Reference Manual*. Cornell University, Hebrew University.

HUTCHINSON, N., & PETERSON, L. 1991. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, **17**(1), 64–76.

JAVAGROUPS. 2004. *JavaGroups Web Site*. http://www.javagroups.com/.

MIRANDA, H., & RODRIGUES, L. 1999. Flexible Communication Support for CSCW Applications. *Pages 338–342 of: 5th Internation Workshop on Groupware - CRIWG'99*. IEEE, Cacún, México.

MIRANDA, H., PINTO, A., & RODRIGUES, L. 2001. Appia, a flexible protocol kernel supporting multiple coordinated channels. *Pages 707–710 of: Proceedings of the 21st International Conference on Distributed Computing Systems*. Phoenix, Arizona: IEEE.

MIRANDA, HUGO. 2001. *Plataforma de suporte ao desenvolvimento e composição de malhas de protocolos*. M.Phil. thesis, Departamento de Informática - Universidade de Lisboa.

PEREIRA, JOSÉ, & OLIVEIRA, RUI. 1997. Object-Oriented Open Implementation of Reliable Communication Protocols. *In: Workshop on Dependable Distributed Object Systems, OOPSLA'97*.

PINTO, A., MIRANDA, H., & RODRIGUES, L. 2001. Light-Weight Groups: an implementation in Ensemble. *In: Fourth European Research Seminar on Advances in Distributed Systems (ERSADS'01)*.

RODRIGUES, L., & GUO, K. 2000. Partitionable Light-Weight Groups. *Pages 38–45 of: Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS'20)*. Taipe, Taiwan: IEEE.

SCHIPER, A., & RICCIARDI, A. 1993. Virtually-Synchronous Communication Based on a Weak Failure Suspector. *Pages 534–543 of: Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*.

SERGIO MENA, XAVIER CUVELLIER, CHRISTOPHE GRÉGOIRE, & ANDRÉ SCHIPER. 2003. Appia *vs.* Cactus: Comparing Protocol Composition Frameworks. *In: 22nd Symposium on Reliable Distributed Systems. Florence, Italy*.

TEIXEIRA, S., VICENTE, P., PINTO, A., MIRANDA, H., RODRIGUES, L., MARTINS, J., & RITO-SILVA, A. 2002. Configuring the Communication Middleware to Support Multi-user Object-Oriented Environments. *Pages 965–980 of: Proceedings of the International Symposium on Distributed Objects and Applications (DOA)*.

VAN RENESSE, R., BIRMAN, KEN, & MAFFEIS, S. 1996. Horus: A Flexible Group Communications System. *Communications of the ACM*, **39**(4), 76–83.