Configuração do Suporte de Comunicação em Ambientes Multi-Utilizador Orientados aos Objectos*

Sandra Teixeira, Pedro Vicente, Alexandre Pinto, Hugo Miranda, Luis Rodrigues

Universidade de Lisboa

{steixeira,pedrofrv,apinto,hmiranda,ler}@di.fc.ul.pt

Jorge Martins, Rito Silva

Instituto de Engenharia de Sistemas e Computadores

{Jorge.B.Martins,Rito.Silva}@inesc.pt

Resumo

Os sistemas distribuídos interactivos multi-utilizador são aplicações que colocam um complexo conjunto de requisitos sobre o suporte de comunicação subjacente. Um modo de satisfazer cabalmente estes requisitos consiste em utilizar arquitecturas de comunicação configuráveis que suportam a reutilização e composição de componentes.

O projecto MOOSCo, "Multi-user Object-Oriented environments with Separation of Concerns" aborda o problema da configuração em sistemas interactivos multi-utilizador. Para suportar a comunicação entre as entidades distribuídas, o projecto recorre ao Appia, uma plataforma de composição de protocolos configurável, que fornece a comunicação em grupo. Este artigo discute o papel do Appia na arquitectura MOOSCo e pretende mostrar como é possível, de uma forma simples e elegante, obter a composição de protocolos mais adequada dependendo dos objectos partilhados pelo ambiente multi-utilizador.

1 Introdução

Aplicações distribuídas tais como os ambientes virtuais, a simulação distribuída, o trabalho cooperativo suportado por computador (CSCW), os jogos multi-utilizador e os MOOs [7] são aplicações que têm ganho uma importância significativa nos últimos anos. Este tipo de aplicações, que pode ser globalmente designado por sistemas distribuídos interactivos multi-utilizador coloca um conjunto vasto de requisitos do ponto de vista da análise, engenharia de software e suporte de sistema.

Devido aos complexos requisitos de fiabilidade, capacidade de escala, adaptabilidade, funcionalidade, mudanças dinâmicas e eficiência deste tipo de sistemas, os ambientes MOO (multi-utilizador orientados aos objectos) constituem um desafio para a teoria e prática dos sistemas distribuídos orientados aos objectos.

O projecto MOOSCo [2] [1], ambientes multi-utilizador orientados aos objectos com separação de facetas (do Inglês, Multi-user Object-Oriented environments with Separation of Concerns), tem como objectivo validar os benefícios de conceber e concretizar este tipo de sistemas usando técnicas de composição e configuração de uma forma vertical e integrada, desde a análise até à implementação. No MOOSCo

*Este projecto foi parcialmente suportado pela Fundação para a Ciência e Tecnologia, através do projecto POSI/CHS/32869/99 MOOSCo.

pretende-se definir e concretizar uma arquitectura que suporte ambientes multi-utilizador orientados aos objectos. A arquitectura apresentada é baseada na composição de componentes e engloba três níveis de abstracção: modelos do utilizador, abstracções intermédias e infra-estrutura dos protocolos de comunicação. Apesar do projecto englobar todas as questões envolvidas no desenvolvimento de MOOs, a interacção entre os objectos, a gestão da percepção, a comunicação distribuída, a partilha de informação, etc, este artigo centra-se na questão da partilha de informação.

Uma forma de abordar a complexidade deste tipo de sistemas consiste em utilizar uma arquitectura configurável, capaz de suportar a reutilização e composição de componentes e uma plataforma de composição de protocolos configurável, denominada *Appia*. Devido às características de composição da arquitectura e da plataforma *Appia* é possível utilizar abstracções intermédias e protocolos de comunicação construídos especificamente para cada modelo de utilizador. A plataforma *Appia* além de fornecer os protocolos de comunicação em grupo ao sistema permite concretizar a partilha de informação de acordo com os critérios do utilizador.

Neste artigo demonstram-se as vantagens do sistema *Appia* como plataforma de composição nos ambientes MOO. Através de um exemplo concreto, ilustra-se como é possível configurar, de forma individualizada, para cada atributo de cada objecto partilhado, um canal de comunicação especializado e como é possível estabelecer relações entre canais de diferentes atributos. Esta configuração é expressa através de um ficheiro em formato XML (eXtensible Markup Language). Deste modo, a aplicação pode ser configurada durante a fase de instalação, não só em função das características intrínsecas dos objectos partilhados, mas também em função das propriedades da infra-estrutura da rede.

Esta artigo está estruturado da seguinte forma: Os diferentes requisitos do sistema MOOSCo são introduzidos na Secção 2. Na Secção 3 é feita uma breve introdução à plataforma *Appia* e descreve-se como esta é utilizada no projecto MOOSCo. Na Secção 5 é apresentado o trabalho relacionado e as vantagens e desvantagens da aproximação apresentada são discutidas na Secção 6. A Secção 7 conclui o artigo.

2 Configuração em sistemas multi-utilizador

2.1 Ambientes multi-utilizador

Os ambientes virtuais multi-utilizador, e em particular os MOOs, são sistemas que suportam a interacção em tempo real de vários utilizadores que podem aceder ao sistema a partir de localizações geograficamente distribuídas. Para este efeito, os MOOs concretizam a noção de espaços virtuais partilhados, frequentemente designados por *salas*, no contexto dos quais os utilizadores podem partilhar dados e informação multimédia (tal como representações gráficas, imagens e sons).

Os MOOs oferecem os mecanismos que permitem aos utilizadores *entrarem* e *saírem* de salas, observarem actividades dos restantes participantes no momento em que estas ocorrem e interagir não só entre si, mas também com o próprio ambiente virtual. Estes sistemas permitem ainda ao utilizador criar e alterar o

conteúdo do ambiente virtual, por alteração, inserção ou remoção de objectos.

Naturalmente, a partilha de informação é um aspecto central na concretização destes ambientes. Existem diferentes tipos de informação que são partilhados pelos utilizadores de um MOO. Em primeiro lugar, os utilizadores estão conscientes da existência dos outros utilizadores que se encontram na mesma sala e, consequentemente, das operações por eles efectuadas. Por exemplo, quando entram ou saem participantes de uma sala, esta informação deve ser propagada por todos. Em segundo lugar, os utilizadores devem ter acesso ao conjunto de objectos localizados na *sala*. Deste modo, a inserção ou remoção de um objecto deve também ser propagada para todos os utilizadores. Para além disso, os utilizadores devem observar as alterações que ocorrem nos objectos. Cada objecto é caracterizado por um ou mais atributos (por exemplo, localização, velocidade, som, etc) que pode ser alterado de forma independente. Finalmente, os requisitos sobre o sistema de comunicação para propagar as alterações a estes atributos podem ser completamente distintos (por exemplo, os protocolos para suportar a propagação de dados são necessariamente diferentes dos utilizados para propagar audio).

Apesar de cada atributo de cada objecto colocar exigências diferentes aos protocolos de comunicação, a propagação da informação referente aos diferentes atributos necessita de respeitar critérios de coerência globais de modo a garantir que todos os participantes têm uma percepção coerente da informação partilhada. Isto quer dizer que o suporte de comunicação para MOOs deve não só permitir configurar as qualidades de serviço adequadas a cada atributo mas também configurar as relações de coerência-mútua entre diferentes atributos.

Mesmo existindo diferentes soluções para o suporte de partilha de informação, poucos sistemas possuem flexibilidade suficiente para permitirem que as aplicações possam adaptar essas soluções aos seus requisitos particulares. Este artigo pretende apresentar uma solução que permite construir a melhor composição de protocolos de acordo com os requisitos da aplicação.

2.2 Soluções monolíticas *versus* soluções configuráveis

No contexto dos MOO, não existe uma solução óptima e única. As soluções devem ser contextuais. A satisfação completa dos requisitos ao nível da adaptabilidade, capacidade de escala, coerência e eficiência destes sistemas não é fácil. Por exemplo, devido à latência da rede, as mensagens podem chegar por ordens diferentes a máquinas diferentes. Isto pode resultar num problema de incoerência, uma vez que diferentes utilizadores recebem diferentes vistas do ambiente. Uma solução *naive* para estes problemas pode recorrer a um servidor centralizado que ordena todas as mensagens. Contudo, devido à sua incapacidade de escala, esta opção limitaria o sistema a um número reduzido de utilizadores. Outra solução pode ser baseada numa pilha de protocolos de comunicação em grupo que forneça ordem total e causal para as mensagens do sistema. No entanto, a utilização destes protocolos para toda a informação trocada no sistema (incluindo audio e video) pode comprometer seriamente o desempenho do sistema, devido ao elevado número de mensagens que este

tipo de protocolos troca. Além de tudo isto, diferentes requisitos da aplicação podem precisar de diferentes níveis de coerência. Não é a melhor opção sobrecarregar o sistema com ordem causal e total quando nem todos os objectos da aplicação necessitam deste tipo de ordenação.

Devido a estes requisitos, o desenho e concretização dos MOOs beneficiará se for utilizada uma aproximação que permita a parametrização de soluções contextuais pela composição de componentes reutilizáveis.

2.3 Requisitos de configuração

Conforme o ambiente virtual e os objectos nele inseridos, o sistema de comunicação para suporte à partilha de dados deve responder a requisitos diferentes. Nesta secção introduzimos um exemplo muito simples que ilustra a complexidade dos requisitos facilmente encontrados neste tipo de aplicações.

Consideremos um jogo simples, tipo MUD (do Inglês, Multi-User Dungeon), em que os jogadores personificam animais que se movem num universo virtual à procura de alimento. Cada participante é representado por um *avatar*, materializado por um objecto partilhado que é caracterizado por dois atributos: a aparência e o movimento. A aparência do avatar reflecte a quantidade de alimento ingerido. No exemplo seguinte, consideramos o caso em que dois participantes interagem num espaço partilhado constituído por uma sala com uma ventoinha e um cesto de frutos. A ventoinha é um objecto cuja posição é pré-definida e que é caracterizada por um único atributo que representa a sua velocidade. O cesto de frutos é caracterizado pela sua posição e pelo número de frutos que contém. Os participantes podem dirigir-se até ao cesto e ingerir um ou mais frutos, o que se deve reflectir na aparência dos seus avatares.

Saliente-se que se os dois participantes tentarem ingerir o último fruto do cesto apenas um deles o deverá conseguir. Deste modo, existe a necessidade de ordenar os movimentos concorrentes que alteram esse atributo. O sistema deve também assegurar que a ordem pela qual as alterações de atributos são observadas respeita a ordem de causa-efeito da acções subjacentes. Por exemplo, a diminuição do número de frutos no cesto deve preceder uma alteração de aparência devido à sua ingestão. Por outro lado, as alterações ao estado da ventoinha são independentes das alterações aos estado dos restantes objectos.

Na discussão que se segue, descrevemos diferentes modos de configurar os protocolos de comunicação numa instalação em que é instanciada uma réplica de cada objecto na aplicação que executa no nó de cada participante. Deste modo, acções que alterem o estado dos atributos devem ser difundidas por todos os participantes.

2.3.1 Canais de comunicação independentes Uma possibilidade de configuração do suporte de comunicação, para a partilha de informação, na sala anteriormente descrita seria utilizar um canal de comunicação independente para cada atributo, tal como se ilustra na Figura 1. Esta arquitectura teria a vantagem de permitir usar qualidades de serviço diferentes para atributos diferentes. Por exemplo, para disseminar alterações à velocidade da ventoinha seria utilizado um canal de difusão fiável com uma ordenação FIFO enquanto

para disseminar os movimentos que podem alterar o estado do cesto seria usado um protocolo assegurando ordem total, de modo a que todos os nós observassem estas acções de forma coerente. A desvantagem desta alternativa é que não é possível, usando canais independentes, assegurar que a entrega das mensagens respeita as relações de ordem causal entre atributos distintos (como entre o número de frutos no cesto e a aparência de quem os ingere). Assim a aparência do avatar poderia mudar antes do número de frutas do cesto diminuir.

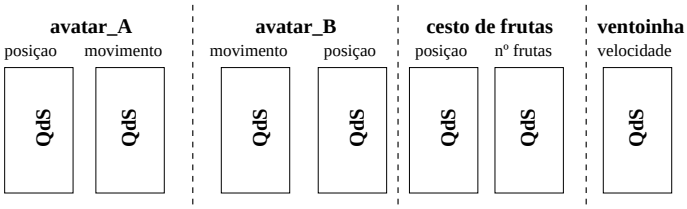

Figura 1: Canais independentes

2.3.2 Um único canal partilhado Uma solução frequente para responder aos problemas de coerência levantados pela arquitectura anterior consiste em usar um único canal que é partilhado por todos os atributos, tal como se ilustra na Figura 2. Este canal partilhado necessitaria de satisfazer a ordenação mais forte requerida pelos objectos partilhados, neste caso ordenação total e causal. A desvantagem desta solução é que a comunicação referente a todos os atributos teria que ser ordenada de forma total, quando apenas um pequeno sub-conjunto destes atributos o exigia. Dado que os protocolos de difusão que asseguram ordem total são significativamente menos eficientes de que os protocolos que asseguram apenas ordem causal, ou mesmo ordem FIFO (para o atributo velocidade da ventoinha), o desempenho global do sistema é bastante penalizado.

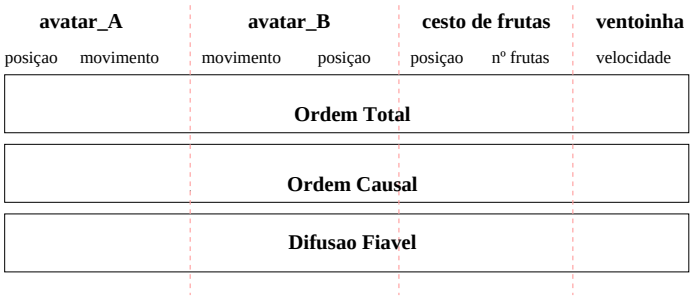

Figura 2: Um único canal partilhado

2.3.3 Canais partilhados com dependências inter-canal Como se acabou de demonstrar, nem a utilização de um canal independente para cada atributo, nem a utilização de um único canal partilhado para todos os atributos satisfaz totalmente os requisitos do nosso exemplo, a primeira opção falha por defeito, a segunda por excesso. A solução passa por permitir criar uma infra-estrutura de comunicação em que alguns atributos possam partilhar algumas propriedades de ordenação sem obrigar a que todos os atributos partilhem essas propriedades.

No nosso exemplo, todos os atributos deveriam utilizar um canal assegurando difusão fiável de informação. Esta qualidade de serviço aliada a uma ordenação FIFO seria suficiente para disseminar a velocidade da ventoinha. Os restantes atributos deveriam partilhar uma ordenação causal comum, dado que podem existir relações de causa-efeito entre a alteração dos valores destes atributos como se referiu anteriormente. Finalmente, o protocolo de ordenação mais forte, a ordem total, deveria ser apenas utilizado para ordenar os movimentos concorrentes dos avatares.

Na próxima secção, descrevemos um sistema de comunicação configurável que permite concretizar este tipo de adaptação.

3 Um sistema de comunicação configurável

3.1 A plataforma de composição *Appia*

O *Appia* [11] é um sistema modular de suporte à comunicação. Cada módulo do Appia é uma camada, *i.e*. um micro-protocolo responsável por garantir uma determinada propriedade. Estas camadas são independentes e podem ser combinadas. Essa combinação constitui uma pilha de protocolos. Esta pilha de protocolos oferece uma Qualidade de Serviço (QdS) com as propriedades desejadas pela aplicação.

Definida uma QdS é possível criar um ou mais *canais* de comunicação que implementam essa QdS. A cada canal está associado uma pilha de *sessões*: uma sessão para cada camada de protocolo cujo objectivo é manter o estado necessário à execução do protocolo da camada correspondente.

A interacção entre camadas é realizada através da troca de eventos que circulam nos canais de comunicação. Os eventos são tipificados e cada camada declara ao sistema quais os eventos que cria e que está interessada em processar. O sistema optimiza o fluxo de eventos na pilha de protocolos, assegurando que os eventos só são entregues às camadas que registaram interesse no seu processamento.

Na secção anterior foram identificados alguns requisitos dos MOOs que podem ser modelados como relações entre canais (por exemplo, a necessidade de existir ordem causal entre diferentes canais). Exemplos semelhantes foram identificados por outros grupos de investigação. O trabalho do CCTL [14] também usa diferentes canais de comunicação que são geridos por um único controlador. O trabalho com o Maestro [5] mostra as dificuldades de manter uma detecção de falhas coerente quando canais com diferentes características são usados simultaneamente. O *Appia* fornece um modelo de composição em pilha que ultrapassa estes problemas, uma vez que permite expressar interdependências entre canais. O conceito que

suporta esta forma de composição é a noção de *sessão partilhada*. Dois canais que partilhem um dado protocolo, podem partilhar um sessão. Dado que é a sessão quem mantém o estado referente à execução do protocolo, a partilha de uma sessão permite que o protocolo correlacione os eventos trocados nos diversos canais. Por exemplo, se vários canais utilizarem uma mesma sessão de um protocolo causal, todas as mensagens trocadas nesses canais serão ordenadas de forma causal entre si.

3.2 Definir canais com sessões partilhadas

Como foi referido na Secção 2.3, os ambientes virtuais partilham um conjunto de objectos e os seus atributos pelos utilizadores. Quando um dos atributos é alterado, o seu novo estado deve ser propagado para os utilizadores. Esta propagação deve respeitar o critério que a aplicação define para este atributo. Com o *Appia*, este critério é projectado num canal de comunicação que oferece a qualidade de serviço pretendida.

Para cada atributo, e de acordo com o pretendido pela aplicação, é criada uma pilha de camadas constituída pelos protocolos necessários para obter a qualidade de serviço escolhida. Sobre cada uma destas camadas é instanciada uma sessão que será utilizada pelo canal que o atributo utilizará para propagar e receber as suas actualizações.

Todos os canais definidos na aplicação, para envio e recepção de actualizações, irão partilhar, na base da sua pilha, as camadas de comunicação fiável em grupo que permitirão enviar as actualizações para os restantes utilizadores de forma a garantir um estado coerente do ambiente. O *Appia* permite também sessões partilhadas, ou seja, que uma sessão lide com mais do que um canal. Esta característica permite, por exemplo, ordenar as mensagens de actualização de dois atributos com uma ordem total. Assim, em vez de para cada camada da pilha ser instanciada uma nova sessão, poderão existir canais que partilham entre eles a mesma sessão de determinada camada.

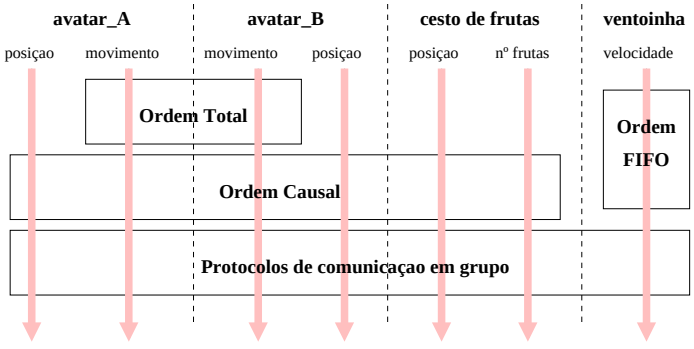

Figura 3: Canais com dependências (sessões partilhadas)

A aplicação deste modelo ao exemplo anterior resulta na configuração representada na Figura 3. Cada atributo possui um canal de comunicação próprio para difundir as suas alterações. No entanto, estes canais não são totalmente independentes. Em particular, todos os canais partilham as sessões do conjunto de pro-

tocolos que assegura a comunicação em grupo fiável. O canal utilizado para disseminar a velocidade da ventoinha possui apenas um protocolo de ordenação FIFO com uma sessão própria. Todos os restantes canais possuem uma camada de ordem causal, partilhando uma sessão comum. Finalmente, os canais utilizados para disseminar os movimentos do avatar possuem ainda uma camada de ordem total, também esta recorrendo a uma sessão partilhada.

3.3 Processo de configuração

Na secção anterior pretendeu-se mostrar como é que a aplicação consegue construir, de acordo com os critérios de coerência definidos pelo utilizador, a pilha de protocolos e respectivos canais necessários para as actualizações dos atributos. Falta agora mostrar como é que o utilizador descreve à aplicação os critérios que pretende, de forma a construir a configuração da Figura 3.

```
<!ELEMENT configuração  (atributo+)>
<!ATTLIST configuração objecto CDATA #REQUIRED>

<!ELEMENT atributo  (nome, QdS)>
<!ELEMENT QdS  (micro-protocolo+)>
<!ELEMENT micro-protocolo  (nome, nomeSessão, modo)>
<!ELEMENT nome  (#PCDATA)>
<!ELEMENT nomeSessão  (#PCDATA)>
<!ELEMENT modo  (#PCDATA)>
```

Figura 4: Gramática de configuração XML

Para ultrapassar esta questão, a linguagem XML revelou-se a melhor opção, uma vez que permite definir um conjunto de regras/convenções para estruturar dados em ficheiros. Estas regras são primeiro definidas numa gramática (DTD - Document Type Definition) que deve ser seguida na criação dos ficheiros XML. Deste modo, estes ficheiros são facilmente especificados e compreendidos devido à sua organização.

O utilizador define então para cada objecto que pretende criar, um ficheiro XML de configuração, que deverá estar de acordo com as regras da gramática que o sistema está preparado para tratar, Figura 4. Este ficheiro, de acordo com a especificação da gramática, deve ter: o nome do objecto ao qual se refere, o nome do atributo em causa, o critério de consistência, denominado QdS - Qualidade de Serviço, onde está descrita a pilha de micro-protocolos e as suas sessões. Os micro-protocolos são identificados pelo seu nome, o qual permite na aplicação criar uma instância nova, ou usar uma já criada deste protocolo. Para cada micro-protocolo, é definida a sessão que o atributo vai usar.

A sessão é identificada por um nome que deverá ser igual nos atributos que pretendem usar a mesma sessão do mesmo protocolo. Além do nome, a sessão tem também o atributo designado por `modo` que tem o seguinte significado: se a sessão não puder ser partilhada, então `modo` deverá ser `local`, se a sessão for partilhada então `modo` deverá ser `partilhado`, e no caso de existir apenas uma sessão para o protocolo em causa então `modo` deverá ser `global`. Assim, se um atributo pretender partilhar a mesma sessão que

um outro atributo já usa, deve indicar o nome da sessão que o outro atributo já utiliza e o nome que este definiu, e o `modo` da sessão deverá estar como `partilhado` ou `global` em ambos os atributos. Se determinada sessão de um protocolo estiver definida como `global`, nenhum atributo poderá pedir uma nova sessão deste atributo, porque a que já existe deve ser partilhada por todos os atributos que usam este protocolo. Do mesmo modo, uma sessão definida como `local`, só pode ser usada por um único atributo.

A Figura 5 mostra os dois ficheiros XML para o exemplo anteriormente descrito. Nestes só são considerados os objectos avatar_A, idêntico para o avatar_B, e a ventoinha uma vez que são suficientes para o exemplo. É possível ver que cada atributo é identificado por um nome, e a sua QdS é composta por micro-protocolos. A ordem como os micro-protocolos são inseridos é importante, uma vez que é por esta ordem que a pilha será construída.

```
<configuração objecto="avatar_A">
 <atributo>
  <nome>movimento</nome>
  <QdS>
   <micro-protocolo>
    <nome>Ordem Total</nome>
    <nomeSessão>sessão total</nomeSessão>
    <modo>partilhado</modo>
   </micro-protocolo>
   <micro-protocolo>
    <nome>Ordem Causal</nome>
    <nomeSessão>sessão causal</nomeSessão>
    <modo>partilhado</modo>
   </micro-protocolo>
   <micro-protocolo>
    <nome>Protocolos de comunicacao em grupo</nome>
    <nomeSessão>sessão grupo</nomeSessão>
    <modo>global</modo>
   </micro-protocolo>
  </QdS>
 </atributo>
 <atributo>
  <nome>posição</nome>
  <QdS>
   <micro-protocolo>
    <nome>Ordem Causal</nome>
    <nomeSessão>sessão causal</nomeSessão>
    <modo>partilhado</modo>
   </micro-protocolo>
   <micro-protocolo>
    <nome>Protocolos de comunicacao em grupo</nome>
    <nomeSessão>sessão grupo</nomeSessão>
    <modo>global</modo>
   </micro-protocolo>
  </QdS>
 </atributo>
</configuração>


<configuração objecto="ventoinha">
 <atributo>
  <nome>velocidade</nome>
  <QdS>
   <micro-protocolo>
    <nome>Ordem Fifo</nome>
    <nomeSessão>sessão fifo</nomeSessão>
    <modo>local</modo>
   </micro-protocolo>
   <micro-protocolo>
    <nome>Protocolos de comunicacao em grupo</nome>
    <nomeSessão>sessão grupo</nomeSessão>
    <modo>global</modo>
   </micro-protocolo>
  </QdS>
 </atributo>
</configuração>
```

Figura 5: Ficheiros de configuração XML

A aplicação consulta estes ficheiros de configuração quando os objectos são inseridos pelo utilizador no ambiente virtual. Cada objecto é propagado para os restantes utilizadores juntamente com o seu ficheiro de configuração. Assim, nos restantes utilizadores, a aplicação remota terá também acesso ao critério especificado pelo utilizador.

	Canais Independentes	Canal partilhado	Canais com dependências
posição	68	145	74
movimento	71	150	147
Coerência Mútua	NÃO	SIM	SIM

Figura 6: Desempenho comparativo

4 Desempenho

O sistema anteriormente descrito foi concretizado através de um conjunto de extensões e novas camadas para o sistema *Appia* [11] e desenvolvendo um protótipo de uma aplicação cooperativa multi-utilizador que permite demonstrar a exequibilidade dos conceitos propostos.

De modo a ilustrar o desempenho do sistema, foi medido o *round trip* das alterações de estado de atributos partilhados. Consideraram-se os atributos "*posição*" e "*movimento*", uma vez que estes têm requisitos diferentes em termos de qualidade de serviço. Para permitir uma análise comparativa, foram medidos três configurações possíveis para os canais de comunicação: *i)* canais independentes, onde cada canal concretiza diferentes propriedades mas sobre os quais não é possível aplicar critérios de coerência mútua; *ii)* um único canal partilhado, usando a qualidade de serviço mais forte requirida pelos dois atributos (total); *iii)* canais com propriedades distintas mas com dependências expressas através de sessões partilhadas.

Os resultados são apresentados na Figura 6 (valores em milisegundos). Como se pode observar, a configuração com canais independentes apresenta um menor *round-trip*, mas não permite coerência entre atributos. Entre as configurações com um canal partilhado e com canais com dependências, a segunda é a mais vantajosa, uma vez que não obriga a utilizar ordem total na disseminação da posição. Pode-se também observar que, nesta última configuração, existe uma penalização no *round-trip* das actualizações da posição em relação à configuração com canais independentes mas pouco significativa, quando comparada com a utilização de um único canal para todos os atributos.

5 Trabalho relacionado

Na área das aplicações interactivas multi-utilizador, sistemas como o AVIARY[7], MASSIVE [12] e o DIVE [8] falham da perspectiva da engenharia de software por não satisfazerem os requisitos de composição, reutilização e parametrização impostos pelo desenho e concretização de ambientes MOO. Como resultado de uma estrutura monolítica, estes sistemas estão limitados a um único modelo de utilização e a um conjunto restrito de abstracções intermédias. Esta limitação também se verifica em ambientes mais recentes orientados aos jogos multi-utilizador [15] [6].

Uma característica comum a todos estes sistemas é o suporte para a partilha de informação. O DIVE [8] e

o SPLINE [3] usam uma base de dados replicada. A interacção entre aplicações e utilizadores é feita através desta base de dados. Apesar destes sistemas oferecerem uma separação clara entre a aplicação e a base de dados replicada, a aplicação tem muito pouco controlo sobre a replicação. Algoritmos de extrapolação (no Inglês, *dead-reckon*) que permitem fazer uma previsão de estado, são utilizados para reduzir as mensagens de actualização de posição. Contudo nenhum destes sistemas permite que o programador especifique algoritmos especializados para a sua aplicação concreta. Do ponto de vista do suporte à comunicação, os sistemas MOO existentes estão normalmente ligados a uma única qualidade de serviço.

Por exemplo, o NPSNET [10] usa apenas comunicação não fiável, enquanto o DIVE utiliza comunicação fiável. O SPLINE suporta os dois tipos de comunicação e a ordenação de mensagens apenas para o mesmo objecto. No entanto em algumas situações seria importante ordenar mensagens entre um conjunto de objectos. Nos sistemas existentes, ainda não é possível suportar qualidades de serviço diferentes de acordo com os requisitos específicos de cada aplicação.

Nos últimos anos tem havido um progresso significativo no desenvolvimento de infra-estruturas de comunicação em grupo. Os sistemas mais recentes já oferecem um conjunto importante de facilidades de configuração. Por exemplo, o Horus [13] permite que as pilhas de comunicação sejam alteradas em tempo de execução; o BAST [9] permite de acordo com o padrão utilizado, e para fornecer o mesmo tipo de serviço, a selecção dos protocolos mais adequados; o Coyote [4] permite que a mesma mensagem seja processada em paralelo por diferentes protocolos. Contudo, estes sistemas não possuem mecanismos que concretizem coordenação entre canais. Trabalhos anteriores revelaram esta limitação, ao tentarem utilizar estes novos sistemas em aplicações multi-canal.

Apesar das vantagens destes sistemas, estes não têm sido muito utilizados no desenho dos MOO, provavelmente devido à dificuldade de expressar relações de dependência entre múltiplos canais de comunicação.

6 Discussão

A principal vantagem da arquitectura proposta neste artigo é a possibilidade do utilizador poder definir a pilha de comunicação mais apropriada aos requisitos de coerência da aplicação. Para cada atributo de um objecto, o utilizador define qual a qualidade de serviço do canal de comunicação de actualizações e que relação este deve ter com os restantes canais.

No entanto, existe ainda bastante trabalho futuro pois subsistem diversas limitações. Uma das desvantagens que esta arquitectura apresenta, é o processo de configuração, pois obriga a um espaço de nomes global para as sessões. Quando dois canais precisam de partilhar a mesma sessão, o nome desta tem que ser igual em ambos os ficheiros de configuração. Na prática o utilizador é obrigado a conhecer a configuração de todos os canais de comunicação para poder estabelecer relações entre canais. A utilização de sessões locais e globais resolve esta limitação, mas apenas em algumas políticas de partilha.

Outro dos desafios levantados pelo modelo prende-se com o desenvolvimento de protocolos de comunica-

ção multi-canal. Apesar da partilha de sessões estar prevista no sistema *Appia* desde a sua génese, alguns dos protocolos já desenvolvidos não consideram esta possibilidade. Este facto deve-se a duas razões. Uma vez que não é usual o conceito de sessões partilhados, os programadores que concretizam os protocolos não o consideram a menos que lhes seja pedido. Por outro lado, a concretização de um protocolo que aceita vários canais por sessão tem uma complexidade variável. Alguns protocolos, como os protocolos de ordenação FIFO são simples de desenvolver, mas outros são muito mais complexos.

O protocolo de Ordem Total utilizado no ambiente MOO descrito acima, ilustra bem a complexidade de vários canais por sessão. O protocolo é baseado num sequenciador: um dos membros do grupo de réplicas é eleito para dar um número de sequência a todas as mensagens trocadas no grupo. Numa concretização multi-canal, apenas uma sequência de números é utilizada em todos os canais que partilham a mesma sessão. Contudo, o protocolo deve decidir se cria um canal novo para trocar a informação de controlo (como os números de sequência) ou se usa um dos canais já existentes, neste caso que canal deverá escolher.

7 Conclusões

Este artigo apresentou uma arquitectura de comunicação configurável que permite satisfazer os requisitos complexos levantados pelas aplicações interactivas multi-utilizador. Esta arquitectura baseia-se na composição vertical e horizontal de micro protocolos. Ao nível da composição vertical, é possível configurar quais os protocolos a executar nos canais que suportam a difusão das actualizações de cada atributo partilhado. Ao nível da composição horizontal é possível concretizar dependências entre estes canais através da especificação de sessões partilhadas. Esta configuração, com uma granularidade significativamente mais fina que todos os sistemas anteriores, pode ser feita durante a fase de instalação do sistema através de um ficheiro de configuração, não necessitando de estar sedimentada no código. Isto permite adaptar a configuração, não só em função das propriedades dos objectos partilhados mas também do ambiente em que o sistema se executa.

Referências

[1] MOOSCo.Multi-user Object-Oriented environments with Separation of Concerns. Home Page URL:http://www.esw.inesc.pt/moosco/.

[2] Miguel Antunes and Rito Silva. Using separation and composition of concerns to build multiuser virtual environments. In *Proceedings of the 6th International Workshop on Groupware - CRIWG'2000*, Madeira, Portugal, 2000. IEEE.

[3] John Barrus, Richard Waters, and David Anderson. Locales: Supporting Large Multiuser Virtual Environments. In *IEEE Computer Graphics and Applications*, pages 16(6):50–100, Nov. 1996.

[4] Nina T. Bhatti, Matti A. Hiltunen, Richard D. Schlichting, and Wanda Chiu. Coyote: A system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366,

November 1998.

[5] K. Birman, R. Friedman, and M. Hayden. The maestro group manager: A structuring tool for applications with multiple quality of service requirements. Technical report, Cornell University, Ithaca, USA, February 1997.

[6] Eric Cronin, Burton Filstrup, and Anthony Kurc. A distributed multiplayer game server system.

[7] Scott Evans. Building blocks of text-based virtual environments. Technical report, Computer Science University, University of Virginia, April 1993.

[8] Emmanuel Frécon and Marten Stenius. Dive: A Scalable Network Architecture for Distributed Virtual Environments. In *Distributed systems Engineering Journal(Special Issue on Distributed Virtual Environments)*, number Vol. 5, No 3, pages 91–100, September 1998.

[9] B. Garbinato and R. Guerraoui. Flexible protocol composition in Bast. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS-18)*, pages 22–29, Amsterdam, The Netherlands, May 1998. IEEE Computer Society Press.

[10] Michael Macedonia, Michael Zyda, David Pratt, Donald Brutzman, and Paul Barham. Exploiting Reality with Multicast Groups. In *IEEE Computer Graphics and Applications*, pages 15(5):38–45, September 1995.

[11] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 707–710, Phoenix, Arizona, April 2001. IEEE.

[12] James Pubrick and Chris Greenhalg. Extending Locales: Awareness Management in MASSIVE-3. In *URL:http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-3*, September 1999.

[13] Robbert Van Renesse, Kenneth P. Birman, Bradford B. Glade, Katie Guo, Mark Hayden, Takako Hickey, Dalia Malki, Alex Vaysburd, and Werner Vogels. Horus: A flexible group communications system. Technical Report TR95-1500, Cornell University, Computer Science Department, March 23, 1995.

[14] I. Rhee, S. Cheung, P. Hutto, and V. Sunderam. Group communication support for distributed collaboration systems. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 43–50, Balitmore, Maryland, USA, May 1997. IEEE.

[15] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. A review on networking and multiplayer computer games. Technical Report 454, Turku Centre for Computer Science, April 2002.