Protocolos em redes de dados

2000-2001

Primeiro trabalho

2 de Outubro de 2000

1 Introdução

O *Appia* [6] é uma plataforma de suporte ao desenvolvimento de protocolos de comunicação que está a ser desenvolvida na FCUL. O *Appia* pretende não só facilitar o desenvolvimento de protocolos mas também a configuração de pilhas de protocolos especializadas que satisfaçam os requisitos específicos das aplicações. A concepção do *Appia* foi inspirada em trabalhos anteriores tais como o x-kernel [3], o Ensemble [2] ou o Coyote [1].

Este trabalho pretende sensibilizar os alunos para a necessidade da construção modular de protocolos de comunicação. Para isso será pedido que cada grupo concretize uma pequena camada de comunicação para o sistema *Appia*.

2 A arquitectura do *Appia*

O *Appia* está a ser construído usando a linguagem Java, e oferece um conjunto de classes e regras para a construção de micro-protocolos, pequenas unidades funcionais que podem ser compostas para criar pilhas de protocolos complexas.

Cada pilha de protocolo pode suportar um ou vários *canais* de comunicação. Cada canal é constituído por uma sequência ordenada de *sessões*, instâncias de uma determinada camada protocolar. Cada sessão mantém o estado necessário à operação da camada. Por exemplo, uma camada que ordene as mensagens de acordo com o política FIFO necessita de manter um número de sequência. A sequência de camadas associadas a um canal define a *qualidade de serviço do canal*. Uma característica do *Appia* é que é possível criar pilhas de protocolos em que diferentes canais partilham a mesma sessão. No entanto, este aspecto de configuração não será experimentado no presente trabalho.

A comunicação entre sessões é feita através da troca de *eventos*. Os eventos são objectos de descendem da classe Event. Usando herança, é possível criar diferentes classes de eventos. Cada camada protocolo deve indicar ao sistema quais os eventos que gera e quais os eventos que está interessada em processar. O núcleo do *Appia* assegura que só são entregues a uma camada os eventos em que esta está interessada.

Para gerar um evento basta criar o objecto correspondente e invocar o método Event.go. Antes de despoletar o evento, a sessão deve iniciar os atributos do evento. Um atributo comum a todos os eventos é a sua direcção, que indica se o evento deve percorrer a pilha na direcção ascendente ou descendente. Após a invocação do método go, o evento é gerido pelo núcleo do *Appia*. Posteriormente, o evento será entregue à sessão seguinte através da invocação do método handle.

O *Appia* segue um modelo de execução que garante que não existe acesso concorrente ao método handle de uma dada sessão por diversos fios de execução. A API do *Appia* é descrita em [5]. O código do *Appia* pode ser obtido em `http://appia.di.fc.ul.pt`.

3 Protocolos a realizar

São propostos diversos protocolos para concretização. Cada grupo concretizará *apenas* um destes protocolos. A atribuição dos trabalhos aos grupos será feita de modo automático, em função do número do grupo.

Todos os trabalhos pressupõem a existência de uma camada de difusão em grupo que assegure a fiabilidade.

3.1 Ordem total simétrica

Este protocolo deve assegurar que as mensagens são entregues ao destinatário de acordo com as relações de *causalidade potencial*, definida do seguinte modo: num sistema distribuído no qual a informação é trocada exclusivamente através da troca de mensagens, diz-se que uma mensagem m_1 *precede*, ou está *potencialmente relacionada de modo causal* com uma outra mensagem m_2, denotando-se $m_1 \rightarrow m_2$, se e apenas se:

- m_1 e m_2 foram emitidas pelo mesmo processo e m_2 foi enviada após m_1 ou;
- m_1 foi entregue ao emissor de m_2 antes de m_2 ter sido emitida ou;
- existe m_3 tal que $m_1 \to m_3$ e $m_3 \to m_2$.

Para além disso o serviço garante a ordem total, isto é, garante que duas mensagens entregues a um par de processos são entregues pela mesma ordem a ambos.

Para o efeito, o protocolo deve manter um relógio lógico de Lamport [4] cujas regras de actualização estão descritas no artigo anexo. Todas as mensagens são estampilhadas com o valor do relógio do emissor.

Todas as mensagens são enviadas em difusão, usando o protocolo anteriormente descrito. Uma mensagem só pode ser entregue após ter sido recebida uma mensagem com uma estampilha igual ou superior de todos os membros do grupo. Sugere-se que sejam utilizadas as seguintes estruturas de controlo: um vector com a última estampilha entregue de cada emissor, outro vector com o número da estampilha da última mensagem recebida de cada emissor, uma função mínimo que obtém o menor valor deste último vector, assim como uma fila de espera ordenada de acordo com as estampilhas. Mensagens com a mesma estampilha são entregues usando uma ordem determinista. O código recuperar a falha de um nó é opcional.

3.2 Ordem total baseada em sequenciador

Neste algoritmo o elemento mais antigo do grupo é designado por sequenciador. Todos os nós devem manter um registo de qual é o sequenciador, o qual deve ser actualizado sempre que muda a filiação do grupo. As mensagens são marcadas como sendo de DADOS e enviadas para todos os nós com um identificador único. Quando recebidas são mantida numa lista de espera. O nó sequenciador deve enviar uma mensagem do tipo ORDEM para cada mensagem de dados que recebe. As mensagem de ORDEM contêm o identificador único da mensagem de dados e um número de sequência. As mensagens devem ser entregues ao utilizador de acordo com o número de sequência das mensagens de ORDEM. O procedimento de recuperação em caso de falha do sequenciador é opcional.

3.3 Ordem total baseada em testemunho

Existe uma mensagem especial, designada por TESTEMUNHO que circula entre todos os membros do grupo. Associado ao testemunho está um número de sequência. Um nó só pode enviar mensagens de dados quando o testemunho está em seu poder. A cada mensagem de DADOS é atribuído um número de sequência pelo seu emissor. O emissor começa a numerar as mensagens a partir do número que recebeu no TESTEMUNHO. Cada nó envia n mensagens e depois passa o testemunho ao próximo nó do grupo. O procedimento de recuperação em caso de falha do dono do testemunho é opcional.

4 Prazos de entrega

O trabalho deve ser entregue dia 6 de Novembro, acompanhado de um relatório. O relatório deve descrever os procedimentos de recuperação de falhas, mesmo que estes não sejam concretizados.

Referências

[1] N. Bhatti, M. Hiltunen, R. Schlichting, and W. Chiu. Coyote: A system for constructing fine-grain configurable communication services. *ACM Trans. on Computer Systems*, 16(4):321–366, November 1998.

[2] M. Hayden. *The Ensemble System*. PhD thesis, Cornell University, Computer Science Department, 1998.

[3] Norman C. Hutchinson and Larry Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engeneering*, 17(1):64–76, January 1991.

[4] L. Lamport. Time, clocks and the ordering of events in a distributed system. *CACM*, 21(7):558–565, July 1978.

[5] H. Miranda, A. Pinto, and L. Rodrigues. Aplication program interface specification of appia. Technical report, Universidade de LIsboa, Faculdade de Ciências, 2000.

[6] H. Miranda and L. Rodrigues. Flexible communication support for CSCW applications. In *5th Internation Workshop on Groupware - CRIWG'99*, pages 338–342, Cancún, México, September 1999. IEEE.