Um Modelo para a Criação de Serviços Cooperativos

Tese apresentada à Universidade de Trás-os-Montes e Alto Douro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Electrotécnica, realizada sob a orientação do Prof. Eurico Manuel Elias de Morais Carrapatoso, Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto.

José Benjamim Ribeiro da Fonseca

O trabalho desenvolvido no âmbito desta tese foi parcialmente financiado ao abrigo do Programa de Desenvolvimento Educativo para Portugal, PRODEP II, através da Acção 5.2 – Formação Avançada de Docentes do Ensino Superior.

Resumo

Os desenvolvimentos tecnológicos ocorridos nas últimas décadas conduziram a uma utilização crescente dos meios informáticos e telemáticos por utilizadores profissionais e domésticos, motivando o desenvolvimento de variadas aplicações para diversas plataformas computacionais. Durante muito tempo, inúmeros sistemas foram desenvolvidos pelos principais fabricantes, sem grandes preocupações de compatibilidade. Esta disparidade de sistemas motivou a definição de normas de interoperabilidade entre os sistemas já desenvolvidos ou a desenvolver.

Um dos desenvolvimentos tecnológicos que mais revolucionaram as actividades profissionais e de lazer nos últimos anos foi a Internet, nomeadamente a *World Wide Web*. A sua facilidade de utilização e os atractivos que coloca em termos de obtenção de informação, de comércio e de actividades de lazer, levaram a um crescimento exponencial da comunidade de utilizadores, potenciado recentemente pela oferta de acesso móvel à *web*.

No contexto organizacional actual, um dos factores de sucesso é a capacidade de realizar eficazmente trabalho em equipa. Este facto tem despertado o interesse das organizações para as aplicações de trabalho cooperativo suportado por computador. Neste tipo de aplicações assumem particular importância os aspectos da interoperabilidade, da familiarização com as aplicações e do suporte à mobilidade dos utilizadores.

Nesta Tese de Doutoramento propõe-se um modelo de criação de serviços cooperativos suportado em tecnologias *web*, através da utilização da tecnologia de *web services* para a comunicação entre os componentes do sistema e para a construção de um conjunto de serviços cooperativos genéricos capazes de se adaptar a diversas aplicações. Para a sua validação, implementou-se um ambiente de criação de serviços cooperativos baseado no modelo, o qual foi usado para desenvolver um protótipo duma aplicação de edição cooperativa de vídeo.

Palavras-chave

Trabalho Cooperativo Suportado por Computador, Sistemas Distribuídos, *Web Services*, Bases de Dados de Objectos, Java, Edição de Vídeo

Abstract

The technological advances that took place in the last few decades led to a growing utilisation of computational and communications resources by both residential and professional users, encouraging the development of several kinds of applications for diverse computational platforms. For considerable time, numerous systems were developed by the major software vendors, without significant concerns with compatibility. These disparate systems encouraged the definition of interoperability standards targeted for developed or under development systems.

One of the technological developments that more revolutionized the professional and leisure activities in recent years was the Internet, namely the World Wide Web. Its ease of utilisation and the attraction it encompasses regarding information retrieval, business and leisure activities, led to an exponential growth of the users' community, fostered by the recent availability of mobile access to the web.

In the current organizational context, one success factor is the ability to effectively realize team work. This fact has raised the interest of organizations in applications of computer supported cooperative work. In this kind of applications interoperability issues, application's familiarity and users' mobility support assume significant importance.

This Thesis proposes a model for the creation of cooperative services supported by web technologies, through the utilisation of web services technology for the communication between system components and for building a set of generic cooperative services adaptable to diverse applications. To validate it, an environment for cooperative services creation based on the model was implemented that was then used to develop a prototype of a cooperative video editing tool.

Keywords

Computer Supported Cooperative Work, Distributed Systems, Web Services, Object Databases, Java, Video Editing

À minha Musa e às minhas Musinhas, fontes de inspiração suprema

Agradecimentos

Uma Tese de Doutoramento é o fruto de anos de trabalho e de dedicação do seu autor e do conjunto de pessoas que o ajudaram a criar as condições materiais e psicológicas essenciais à realização dos seus intentos.

À UTAD agradeço as condições de trabalho proporcionadas, entre as quais se inclui a dispensa de serviço docente concedida, importante para a realização desta tese.

Agradeço ao meu orientador, Prof. Eurico Carrapatoso, o seu valioso apoio em todo o processo de orientação desta tese, não esquecendo a dignidade demonstrada nos momentos mais difíceis da actividade profissional e da vida pessoal.

Ao Prof. José Bulas Cruz agradeço o incentivo e o apoio proporcionado para a realização deste trabalho.

Agradeço igualmente ao Hugo Paredes, ao António Marques e ao Paulo Martins a sua amizade e as suas sugestões, que muito contribuíram para a realização do trabalho desenvolvido no âmbito desta tese.

À minha colega Paula Oliveira, agradeço a amizade, o bom ambiente de trabalho proporcionado no espaço que partilhamos e os importantes apoios e incentivos à realização deste trabalho.

De uma forma geral, agradeço aos colegas da UTAD a sua solidariedade, a camaradagem e a partilha de experiências de vida e profissionais, que são sempre enriquecedoras.

Por fim, mas não em último lugar, um agradecimento especial à minha família, pelo infinito apoio, dedicação e paciência que sempre demonstraram. Aos meus pais agradeço o carinho, os valores que me transmitiram e o constante incentivo. Às minhas filhas, Catarina e Margarida, agradeço a sua ternura, o seu apoio e a reconfortante alegria. Finalmente, à minha esposa, Paula, agradeço o seu amor, o seu apoio incondicional e a infinita paciência.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo faz-se um enquadramento do problema e das motivações que conduziram à definição do modelo proposto nesta tese. São também definidos os objectivos que se pretendem atingir ou verificar com o trabalho desenvolvido e apresenta-se a estrutura da tese, referindo resumidamente os assuntos que são tratados em cada um dos capítulos seguintes.

1.1. Enquadramento

Nas últimas décadas assistiu-se a uma proliferação generalizada dos equipamentos informáticos e ao desenvolvimento de aplicações cada vez mais sofisticadas, de modo a satisfazer um universo de utilizadores em constante crescimento e cuja exigência aumenta proporcionalmente à sua familiarização com as novas tecnologias. Este crescimento traduziu-se também na existência de uma grande diversidade de equipamentos e programas, nomeadamente no que diz respeito aos sistemas operativos e às ferramentas de desenvolvimento.

Os equipamentos de comunicação sofreram também importantes evoluções nas últimas décadas, com a generalização das redes informáticas nas organizações, a grande expansão da Internet, mesmo no mercado doméstico, e a adopção massiva das comunicações móveis, cujas gerações mais recentes abrem novas perspectivas de utilização.

As redes locais de computadores generalizaram-se no mercado empresarial, atingindo um grande número de profissionais das mais variadas áreas, mas a sua interligação revelou-se fundamental para o sucesso obtido. A Internet surgiu, assim, como um meio de comunicação de importância crescente nos mercados empresariais e domésticos, interligando sistemas dos mais variados fabricantes e com sistemas operativos diversos.

Entre a grande comunidade de utilizadores da Internet despertou o interesse pelo desenvolvimento de novas aplicações baseadas na *web*, quer ao nível do entretenimento, quer ao nível das aplicações de negócio. O grande óbice a um desenvolvimento mais rápido do número e diversidade de aplicações baseadas na *web* tem sido a dificuldade em colocar sistemas heterogéneos a comunicarem de forma eficiente e fiável. De facto, durante muito tempo, os principais fabricantes de equipamentos e programas informáticos produziram diversas soluções proprietárias, optimizadas para os seus sistemas, e as tentativas de normalização das comunicações em ambientes heterogéneos, motivadas pela deficiente adequação desta abordagem à realidade emergente, não produziram resultados satisfatórios. Nos últimos anos surgiram esforços, por parte da comunidade científica e dos principais fabricantes, no sentido de produzir normas que

permitissem solucionar este problema. Destes esforços, o mais recente, conhecido por *web services*, tem despertado grande interesse e expectativas quanto à capacidade de, num futuro próximo, conseguir proporcionar os níveis de interoperabilidade, fiabilidade, economia e segurança necessários ao amadurecimento dos serviços disponibilizados aos utilizadores através da Internet, sem necessidade de ruptura com os sistemas e protocolos já desenvolvidos.

A eficácia do trabalho em equipa nas empresas e instituições é um dos factores que influenciam marcadamente o seu sucesso. A adopção dos meios informáticos e telemáticos por parte dessas organizações trouxe novos desafios relativamente aos meios que podem ser usados na prossecução das tarefas relativas ao trabalho em grupo, surgindo uma nova área de investigação, a que se dá o nome de trabalho cooperativo suportado por computador (habitualmente designado na terminologia anglo-saxónica por *Computer Supported Cooperative Work* – CSCW). O trabalho cooperativo enquadra-se num domínio multidisciplinar, no qual se podem envolver, para além dos programadores de aplicações, peritos de áreas sócio-profissionais. Mesmo dentro do domínio computacional, enquadram-se as áreas dos sistemas distribuídos, das ciências da informação, das tecnologias multimédia e das comunicações.

Um dos aspectos a ter em consideração no desenvolvimento de aplicações cooperativas é a facilidade com que os utilizadores se poderão adaptar aos novos métodos de trabalho que a sua adopção poderá implicar. Desta forma, torna-se importante estudar previamente os hábitos de trabalho dos utilizadores e avaliar o impacto produzido pela introdução das aplicações cooperativas. Mesmo quando as aplicações cooperativas parecem trazer benefícios óbvios para a organização, nem sempre são bem aceites pelos profissionais, pela simples alteração de hábitos enraizados, razão que pode ditar um insucesso prematuro deste tipo de aplicações. Outro motivo de insucesso pode ser a inadaptação das tecnologias usadas aos objectivos da aplicação. De qualquer forma, o trabalho cooperativo suportado por computador é uma área ainda relativamente jovem e pouco desenvolvida face ao potencial de oportunidades que oferece aos programadores e às organizações.

Uma das características cooperativas mais desejadas num grande número de aplicações é a possibilidade de partilhar áreas de trabalho, nas quais diversos utilizadores podem actuar e ter a percepção das acções tomadas pelos seus parceiros de trabalho. Esta característica coloca desafios quanto à forma como é feito o reflexo das

actividades dos diversos utilizadores e relativamente aos mecanismos necessários para garantir a consistência dos dados partilhados.

Nos diversos domínios de aplicações têm-se desenvolvido essencialmente protótipos, sem grande expressão no que diz respeito à passagem a produtos comerciais. A área onde mais se tem investido é no desenvolvimento de sistemas de coordenação de processos de trabalho (vulgarmente conhecido por *Workflow*), aos quais as grandes organizações têm aderido com algum sucesso, bem como de aplicações de realidade virtual e editores de grupo. Contudo, subsistem ainda muitas áreas onde pouco ou nada foi feito e que são campo de investigação de novas aplicações cooperativas.

Como já foi referido, um forte motivo de rejeição ou fraca aceitação de aplicações cooperativas é a mudança significativa de métodos de trabalho. Mesmo quando já são utilizados meios informáticos e telemáticos para a prossecução das tarefas organizacionais, a adopção de ambientes gráficos e paradigmas de utilização muito diferentes pode conduzir a algum desconforto e rejeição. Uma categoria de ambientes computacionais com a qual a maioria dos profissionais das organizações está familiarizada é a das aplicações *web*. De facto, uma parcela significativa dos profissionais das empresas e das instituições está habituada a utilizar a *web*, no trabalho e em casa, para pesquisas de informação e troca de mensagens ou documentos. Desta forma, poderá ser benéfico adoptar ambientes gráficos semelhantes aos da *web*, no suporte a aplicações cooperativas. Outro aspecto igualmente importante poderá ser o facto de, na mesma organização, existirem diversos ambientes computacionais, introduzindo, mais uma vez, o problema da interoperabilidade, se não se quiser alterar os ambientes de cada utilizador. Revela-se, assim, atractiva a adopção de tecnologias de comunicação e representação da informação que permitam uma interligação transparente dos vários sistemas, como acontece com os *web services*.

A utilização dos *web services* em aplicações de trabalho cooperativo não tem ainda grande expressão, à excepção dos sistemas de *workflow*, onde existe inclusive trabalho de normalização relativamente ao formato das mensagens XML. Desta forma, poderão existir muitas outras aplicações cooperativas desenvolvidas eficazmente recorrendo a *web services*, como suporte à comunicação entre os componentes do sistema e como fornecedores dos serviços de que essas aplicações necessitam.

1.2. Objectivos

Como consequência dos problemas e motivações que foram apresentados na secção anterior, o objectivo primordial desta tese foi o de definir um modelo de criação de serviços de trabalho cooperativo suportado por computador, usando a *web* como meio de comunicação entre as aplicações cooperantes e recorrendo a tecnologias recentes e ainda pouco experimentadas no contexto das aplicações cooperativas. Esse modelo devia ser suficientemente genérico, para que pudesse ser utilizado nas mais variadas aplicações cooperativas, mas devia também ser validado através da implementação de um ambiente de suporte a actividades cooperativas baseado no modelo e de um protótipo de uma aplicação cooperativa suficientemente sofisticada e inovadora. Não se pretendia, contudo, validar a utilização das aplicações do ponto de vista da eficácia organizacional, dado que essa é uma tarefa da esfera da Sociologia.

A concretização do objectivo principal desta tese passava pelo cumprimento de várias etapas, necessárias ao correcto enquadramento tecnológico do problema e à escolha das soluções que parecessem ser as mais adequadas aos pressupostos da definição do modelo. A seguir apresentam-se os objectivos parcelares resultantes do desdobramento do objectivo principal da tese:

- estudo das principais arquitecturas de processamento distribuído e análise das vantagens e desvantagens de cada uma delas, sob vários pontos de vista;
- estudo das diversas categorias de aplicações cooperativas, dos seus requisitos e das suas condicionantes;
- identificação dos principais sistemas cooperativos disponíveis comercialmente ou apresentados nas publicações e reuniões científicas internacionais mais importantes;
- proposta de um modelo genérico para a criação de serviços cooperativos, que comuniquem entre si através da *web*;
- análise das tecnologias que possam servir de suporte à implementação de um ambiente cooperativo baseado no modelo proposto e selecção das mais adequadas;

- escolha de produtos comerciais ou gratuitos que implementem as tecnologias seleccionadas e que se adequem ao ambiente computacional disponível ou susceptível de ser criado;
- escolha de uma aplicação cooperativa inovadora, que sirva de caso de estudo para o modelo proposto na Tese;
- implementação de um ambiente para a criação de aplicações cooperativas baseado no modelo proposto;
- implementação de um protótipo da aplicação cooperativa escolhida para caso de estudo;
- realização de testes que permitam validar qualitativamente o modelo proposto e, se possível, de uma forma quantitativa;
- definição de linhas de trabalho futuro, à luz dos resultados obtidos com o protótipo implementado e das potencialidades que o modelo apresente e que não tenham sido totalmente exploradas, atendendo à evolução previsível das normas e tecnologias disponíveis.

Uma Tese de Doutoramento deve contribuir com algo de novo para a área de conhecimento em que se integra. As principais contribuições que o trabalho realizado e descrito nesta Tese apresenta são as seguintes:

- verificação da viabilidade de utilização de *web services* como suporte a serviços cooperativos, com características distintas das dos serviços de negócio, que têm sido a aplicação preferencial desta tecnologia;
- definição e construção de um conjunto básico de serviços de suporte a aplicações cooperativas, suficientemente genéricos para que se possam aplicar a uma gama alargada de situações;
- construção de um protótipo de uma aplicação cooperativa inovadora, visto que não foram encontradas implementações comerciais, nem ecos de actividade significativa de desenvolvimento nessa área.

1.3. Estrutura da tese

Esta Tese de Doutoramento está estruturada em seis capítulos, os quais reflectem aproximadamente o conjunto de assuntos identificados na definição de objectivos feita anteriormente.

A seguir a este capítulo, de introdução, o capítulo 2, *Arquitecturas de Processamento Distribuído*, apresenta uma caracterização genérica dos sistemas distribuídos, debruçando-se sobre os principais modelos de aplicações distribuídas, sobre os seus principais requisitos, vantagens e desvantagens, e descrevendo resumidamente as principais arquitecturas de processamento distribuído da actualidade: RMI, DCOM, CORBA e *Web Services*. Destas, dá-se alguma ênfase às duas últimas, por serem as que assumem maior importância no contexto actual dos sistemas distribuídos, terminando o capítulo com uma comparação das principais características das quatro arquitecturas de processamento distribuído.

No capítulo 3, *Trabalho Cooperativo Suportado por Computador*, descrevem-se aspectos genéricos relacionados com o trabalho cooperativo suportado por computador. Para além das definições de termos associados ao trabalho cooperativo suportado por computador, apresentam-se os principais métodos de classificação de aplicações nesta área, bem como os seus domínios de aplicação. É feita igualmente uma análise dos requisitos e desafios que o projecto e desenvolvimento deste tipo de aplicações apresentam, bem como das suas vantagens e desvantagens. Faz-se também uma descrição das diversas classes de aplicações cooperativas e referem-se algumas implementações conhecidas, finalizando-se com uma breve discussão acerca da utilização da *web* para o seu suporte.

O capítulo 4, *Um Modelo para a Criação de Serviços Cooperativos*, apresenta o modelo proposto nesta Tese, referindo as motivações que conduziram ao seu desenvolvimento e descrevendo a sua arquitectura genérica e, mais detalhadamente, as componentes dessa arquitectura. Termina-se o capítulo com uma discussão de possíveis cenários de aplicação do modelo.

É apresentado no capítulo 5, *Um Ambiente Cooperativo baseado no Modelo de Criação de Serviços Cooperativos*, a implementação de um ambiente para a criação de aplicações cooperativas baseado no modelo proposto no capítulo 4, sendo discutidas as

opções de implementação disponíveis e as razões para as escolhas efectuadas. Este ambiente implementa as características que foram consideradas importantes para a validação do modelo e para suportar a aplicação escolhida para caso de estudo. A aplicação escolhida foi uma aplicação simplificada de edição cooperativa de vídeo, tema para o qual não se encontraram implementações cooperativas disponíveis nem referenciadas. O capítulo 5 termina com uma breve análise do desempenho do ambiente cooperativo implementado, baseada em observações qualitativas e em algumas medições temporais.

Finalmente, no capítulo 6, *Conclusões*, apresentam-se algumas conclusões relativas aos resultados obtidos e aos objectivos propostos e apresentam-se as linhas de evolução futura que o modelo e os protótipos apresentados permitem perspectivar.

Capítulo 2

Arquitecturas de Processamento Distribuído

Neste capítulo, começa-se por apresentar uma caracterização genérica dos sistemas distribuídos, focando os seus principais requisitos, problemas e vantagens, bem como os modelos de aplicações distribuídas. Seguidamente, descrevem-se resumidamente as arquitecturas mais importantes da actualidade, dando-se algum destaque às arquitecturas CORBA e de *Web Services*, visto serem as que nos últimos anos despertaram maior interesse e maiores expectativas na comunidade científica. Finalmente, encerra-se o capítulo com a comparação das principais arquitecturas de processamento distribuído.

2.1. Introdução

Até meados da década de 80, a maioria dos sistemas informáticos eram constituídos por computadores de grande porte e de preço elevado. Como resultado disto, muitas organizações possuíam apenas um ou, quando muito, alguns computadores que operavam, geralmente, isolados.

Durante a década de 80, dois avanços tecnológicos contribuíram decisivamente para que esta situação se alterasse. Um desses avanços foi o aparecimento de microcomputadores com processadores (**CPU** - *Central Processing Unit*) mais rápidos, de 16 e 32 bits (e de 64 bits, já na década de 90), em contraste com os antigos CPUs de 8 bits. O outro avanço tecnológico decisivo foi o aparecimento das redes locais (**LAN** - *Local Area Network*), nomeadamente das redes *Ethernet*, que permitem a interligação de centenas de máquinas a cadências de transmissão relativamente elevadas. Estes dois avanços tecnológicos tornaram possível interligar, através de redes de alta velocidade, sistemas computacionais constituídos por um grande número de CPUs. Estes sistemas são habitualmente designados por **sistemas distribuídos**, em contraste com os anteriores sistemas centralizados, constituídos por um só CPU, memória, alguns periféricos e alguns terminais [1].

Mais recentemente, na década de 90, a Internet conheceu uma enorme evolução, quer no que diz respeito à quantidade de utilizadores (passou de 188.000 computadores conectados em 1990 para 44.000.000 em 1999), impulsionada pelo desenvolvimento e proliferação dos computadores pessoais e das redes de computadores, quer relativamente à quantidade, qualidade e diversidade de serviços disponibilizados.

Durante décadas, os fabricantes de equipamento informático desenvolviam as suas próprias soluções de *software* para as tarefas mais importantes, motivados por um melhor conhecimento das potencialidades e especificidades das suas máquinas. A consequência desta estratégia foi uma progressiva dependência do cliente em relação ao fabricante e as dificuldades de interoperabilidade entre os diferentes sistemas. Contudo, a evolução da indústria de *hardware* e de *software* na década de 90, bem como a Internet, tornaram essa abordagem cada vez menos eficaz, o que levou ao desenvolvimento de esforços de normalização na área dos sistemas distribuídos abertos, realizados por uma parte significativa das empresas produtoras de *software*. O resultado

desse esforço foi primeiro a arquitectura CORBA, para a qual existe já uma grande variedade de implementações, e mais recentemente os *web services*, que prometem ainda um maior grau de interoperabilidade. Simultaneamente, surgiram algumas arquitecturas proprietárias com ambições de se tornarem normas *de facto*, nomeadamente a JavaRMI e a DCOM. Apesar de não existir ainda um desfecho conclusivo acerca de qual das arquitecturas vingará, a disputa mantém-se essencialmente entre a arquitectura CORBA e os *web services*.

2.2. Requisitos

Uma aplicação distribuída deve ter um comportamento semelhante ao que teria a sua equivalente centralizada. De facto, o utilizador não se deve preocupar com as especificidades da distribuição da aplicação. A esta propriedade dá-se o nome de **transparência** na utilização. A transparência deve existir também do ponto de vista do programador, podendo ocorrer em relação a vários aspectos, dos quais se destacam os seguintes:

- **localização** - o programador não necessita de conhecer a localização exacta de uma determinada aplicação, precisando apenas de conhecer o seu nome simbólico;
- **acesso** - as diferenças na representação de dados entre sistemas distintos devem ser ocultadas;
- **migração** - uma aplicação pode migrar de uma máquina para outra, sem o programador se preocupar com esse aspecto;
- **replicação** - várias instâncias de um serviço devem ser vistas como uma única;
- **concorrência** - o programador não precisa de se preocupar com mecanismos que permitam a execução concorrente de pedidos de serviços;
- **sincronização e ordenamento de actividades** - o programador não necessita de especificar nem conhecer a ordem pela qual as actividades são executadas.

Outro requisito importante na óptica do utilizador é a **partilha de informação**, a qual, por sua vez, exige um maior cuidado com os aspectos de **segurança**, **fiabilidade** e **disponibilidade**.

A localização das componentes de uma aplicação distribuída em máquinas distintas exige que a arquitectura de processamento distribuído suporte diversas plataformas, ou seja, exige o suporte de **interoperabilidade** entre sistemas. Esta interoperabilidade deve englobar os sistemas operativos, as redes e os protocolos de transporte.

Os sistemas distribuídos devem contemplar também os requisitos de **escalabilidade.** De facto, um problema que cada vez mais se coloca aos programadores e gestores de sistemas informáticos é o da evolução dos sistemas de uma forma que permita rentabilizar os investimentos em equipamento, em *software* e na formação de técnicos e utilizadores [2].

2.3. Problemas

O maior problema dos sistemas distribuídos é a sua programação, pois este tipo de sistemas necessita de programas substancialmente diferentes dos programas construídos para sistemas centralizados. Os programadores de aplicações distribuídas devem tentar fazer com que cada aplicação se comporte, tanto quanto possível, como a versão centralizada do mesmo programa. Essencialmente, o objectivo da computação distribuída é esconder a localização geográfica dos computadores e dos serviços, fazendo com que eles pareçam locais [3].

Apesar de todos os esforços desenvolvidos para simplificar a programação de aplicações distribuídas, esta continua ainda a caracterizar-se por um maior grau de **complexidade** em relação às aplicações centralizadas. Por outro lado, a tentativa de ocultação de determinados detalhes relativos à distribuição, introduz diversas camadas protocolares, as quais vão necessariamente afectar o **desempenho** da aplicação. Para além disso, a própria comunicação com sistemas remotos pode afectar o desempenho da aplicação de uma forma perceptível, dependendo obviamente da localização do sistema remoto e da rede subjacente.

O desempenho pode ser avaliado usando vários critérios, tais como a resposta temporal e a eficiência. A resposta temporal é afectada pelo processo de comunicação, pois o envio de uma mensagem e a espera pela sua resposta podem demorar um tempo considerável, devido, principalmente, ao tempo gasto pelos protocolos de comunicação. Assim, para optimizar o tempo de resposta é preciso minimizar o número de mensagens trocadas; isto conduz a um dilema, pois a melhor maneira de incrementar o desempenho de um sistema é colocar várias actividades a executar em paralelo em vários processadores, mas tal comportamento requer que sejam enviadas mais mensagens. De uma forma geral, nos trabalhos que envolvem uma grande quantidade de pequenas computações não compensa a utilização de invocações remotas, pois os atrasos sofridos no processo de comunicação não compensam o potencial ganho em tempo de processamento, para além de poderem conduzir a uma mais rápida saturação da rede. Em contrapartida, nos trabalhos compostos por algumas computações de grande porte, pode ser vantajosa a utilização de processamento paralelo.

A partilha de informação entre as componentes de uma aplicação distribuída conduz necessariamente a um aumento do potencial de **insegurança** no acesso à informação armazenada.

Outro problema inerente aos sistemas distribuídos, que já aqui foi abordado, é a **interoperabilidade** entre sistemas, devido à heterogeneidade de equipamentos, redes e programas informáticos. Esta heterogeneidade tem uma expressão significativa, por exemplo, na representação de dados, onde os *bytes* constituintes de um número inteiro não são representados pela mesma ordem.

2.4. Vantagens

Uma vantagem facilmente verificável diz respeito ao facto de as organizações dispersas geograficamente terem necessidade de distribuir a totalidade ou parte do seu sistema de informação.

A **redução de custos** é outra das potenciais vantagens dos sistemas distribuídos sobre os sistemas centralizados. De facto, o crescente incremento do poder computacional dos computadores pessoais, aliado à diminuição do seu preço, permite a

construção de sistemas que, distribuindo adequadamente o processamento, disponibilizem um poder computacional global semelhante ou até superior ao dos *mainframes*, mas a um preço inferior.

Alguns dos problemas colocados à programação de sistemas distribuídos podem, em certos casos, transformar-se em vantagens.

A distribuição do poder computacional por diversos processadores poderá também ser uma vantagem no que diz respeito ao próprio **desempenho**, potenciado pelo processamento paralelo de tarefas.

A possibilidade de replicação de servidores de aplicações pode aumentar a **fiabilidade** e **disponibilidade** dos sistemas distribuídos, permitindo que, em caso de falha de um servidor, este seja substituído por uma réplica. Para além disso, a **flexibilidade** proporcionada pela replicação permite efectuar um constante balanceamento da carga computacional, evitando o mais possível a saturação dos servidores.

Os sistemas distribuídos vêm o seu **crescimento progressivo** facilitado pela aquisição de processadores extra e pela sua adição ao sistema. Para além disso, esse crescimento pode ser obtido sem necessidade de descartar os sistemas informáticos mais antigos, permitindo a sua coexistência com os sistemas recém-adquiridos.

2.5. Modelos de comunicação distribuída

A comunicação distribuída surgiu como uma evolução da comunicação entre processos residentes numa mesma máquina. Numa primeira aproximação, o controlo dos mecanismos de comunicação era efectuado directamente pelo programador das aplicações, o qual tinha que lidar com todos os problemas inerentes, já apontados anteriormente. A evolução dos sistemas operativos e das necessidades de comunicação distribuída levou a que esta se passasse a efectuar por intermédio de protocolos de comunicação implementados ao nível do núcleo do sistema operativo, existindo uma interface de programação (**API -** *Application Programming Interface*) que abstrai grande parte dos detalhes relativos a esses protocolos, conforme se encontra ilustrado na Figura 2-1.

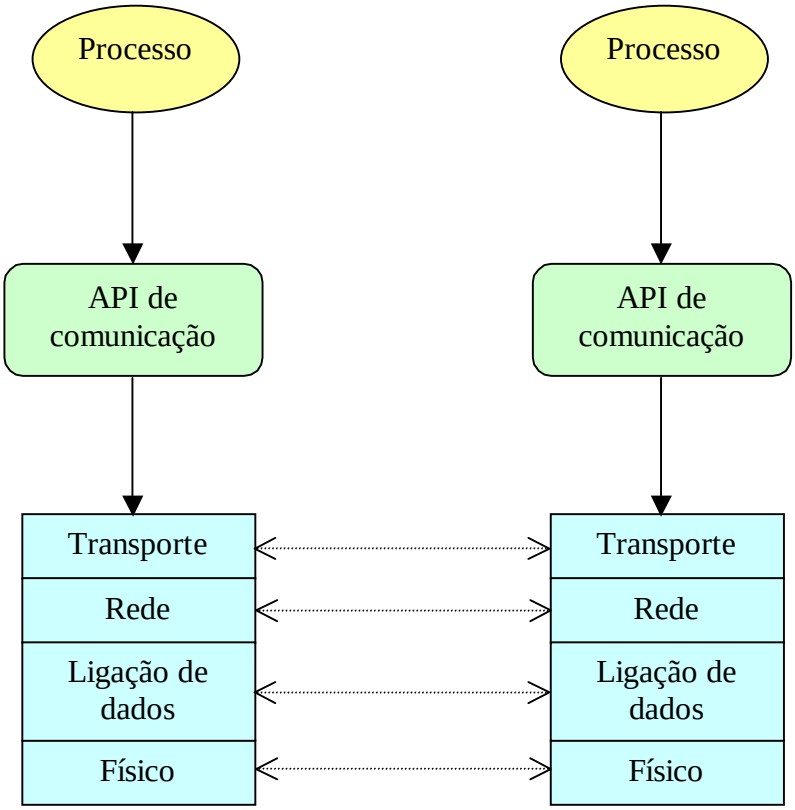

Figura 2-1– API de comunicação distribuída

Para que se possa efectuar a troca de informação entre os dois processos, tem que se estabelecer um **canal de comunicação** entre eles. Nas extremidades do canal encontram-se os pontos de acesso ao mesmo, designados **portos**, os quais fazem a interface entre o sistema operativo e o protocolo de transporte. Existem dois tipos de protocolos de comunicação:

- protocolo orientado à conexão (*connection oriented*) – há uma alocação de recursos de transmissão entre os dois processos. Permite a bidireccionalidade, é fiável e garante a sequencialidade da entrega das mensagens, mas possui uma latência inicial para o estabelecimento do canal que o torna mais adequado para fluxos contínuos de informação;
- protocolo sem conexão (*connectionless*) – não há uma alocação fixa de recursos de transmissão. Não há garantias de sequencialidade e é menos fiável que o protocolo orientado à conexão, mas a latência inicial é

reduzida, mostrando-se adequado para a troca de mensagens curtas, eventualmente entre vários interlocutores.

2.5.1. Modelo cliente-servidor

O propósito do modelo cliente-servidor é a estruturação dos sistemas como grupos de processos cooperantes, onde existem processos, designados **servidores**, que oferecem serviços a outros processos, designados **clientes**. Numa determinada máquina podem existir vários clientes, vários servidores ou ambos, como pode ser visto na Figura 2-2.

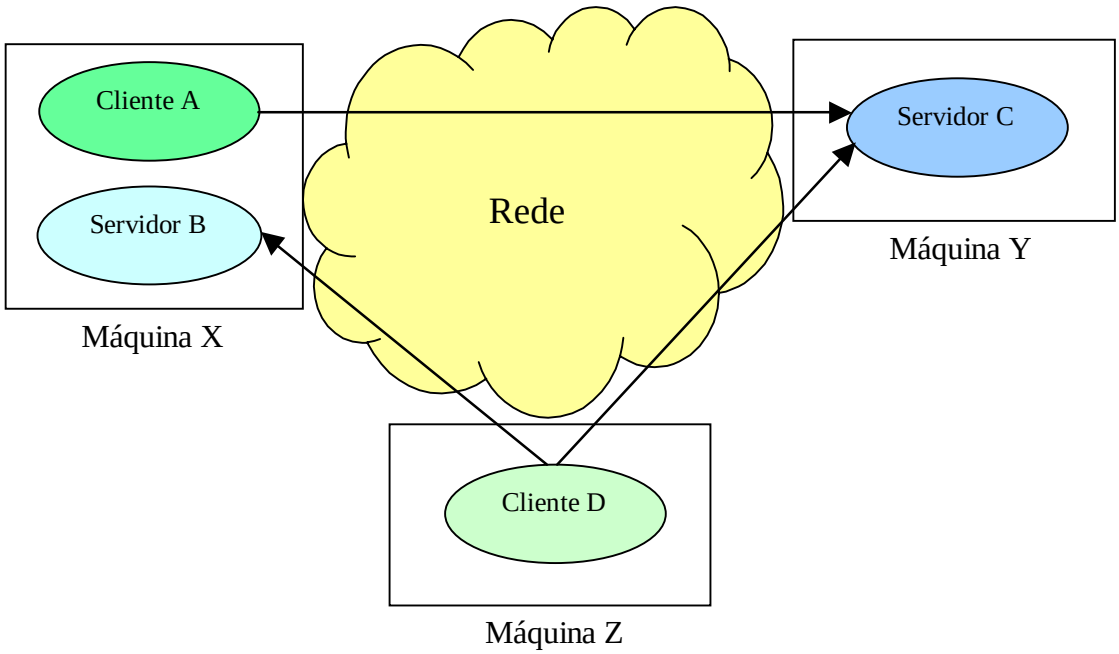

Figura 2-2 – Modelo Cliente/Servidor

Para eliminar a elevada latência inicial dos protocolos orientados à conexão, o modelo cliente-servidor utiliza habitualmente um protocolo sem conexão simples, baseado em mensagens: o cliente envia uma mensagem ao servidor pedindo um determinado serviço; o servidor faz o trabalho pretendido e retorna uma mensagem contendo o resultado obtido, ou uma mensagem de erro, se o trabalho não for executado com sucesso.

Se o cliente ficar bloqueado enquanto o servidor processa o seu pedido, estamos perante um **mecanismo síncrono** de passagem de mensagens. Num **mecanismo assíncrono**, enquanto o servidor processa o pedido, o cliente fica livre para executar em paralelo outra actividade qualquer, até que seja notificado para receber a resposta ao pedido feito ao servidor.

Sockets

Um dos exemplos mais conhecidos do modelo cliente/servidor é a interface de programação de ***sockets***, usada para construir aplicações tão populares como o `telnet` e o `ftp`. A API de sockets fornece um conjunto de métodos que permitem estabelecer canais orientados à conexão e canais sem conexão, efectuar a transferência de informação nesses canais e proceder à sua libertação quando já não forem necessários. Apesar de fornecer já alguma abstracção protocolar, continua a ser necessário controlar explicitamente diversos aspectos da comunicação, por intermédio da invocação dos métodos da API.

RPC

A comunicação implementada utilizando a API de *sockets* é baseada em mecanismos de troca de mensagens. O objectivo dos sistemas distribuídos é fazer com que estes pareçam centralizados, mas o facto de a comunicação por troca de mensagens não ser uma característica dos sistemas centralizados faz com que não seja conveniente para sistemas distribuídos. De facto a semântica de comunicação dentro dos sistemas centralizados é essencialmente procedimental.

Uma solução é permitir que os programas possam invocar procedimentos localizados noutras máquinas [4]. Quando um processo numa máquina A invoca um procedimento numa máquina B, o processo A é suspenso e a execução ocorre em B. A informação pode ser transmitida de A para B através dos parâmetros da invocação, e de B para A através do resultado da invocação. Para o programador não é visível qualquer

mecanismo de passagem de mensagens. Este método é conhecido por chamada de procedimentos remotos ou, mais vulgarmente, **RPC** (*Remote Procedure Call*).

Como os procedimentos invocador e invocado correm em máquinas diferentes, os seus espaços de endereçamento são diferentes, o que causa problemas. A passagem de parâmetros e resultados é outro problema, por causa dos formatos de representação dos dados em máquinas diferentes.

Quando um cliente deseja executar uma operação remota, os parâmetros não são colocados em registos, como acontece nas invocações locais. Em vez disso, os parâmetros são empacotados (***marshalling***) numa mensagem que o *Kernel* envia ao servidor. O componente do cliente que efectua esse empacotamento, bem como o desempacotamento (***unmarshalling***) dos resultados recebidos, é o ***stub* do cliente**. Da mesma maneira e do lado do servidor temos o ***stub* do servidor**, que faz o empacotamento e desempacotamento das mensagens com que lida. No processo de empacotamento de parâmetros é feita a tradução dos dados, representados num determinado formato, para um formato canónico de rede. No desempacotamento de parâmetros é feita a operação inversa, ou seja, a tradução dos dados representados na forma canónica para o formato da máquina que os recebe.

Após fazer uma invocação, o cliente bloqueia, ficando à espera dos resultados; da mesma forma, quando o servidor não está a atender nenhum pedido, fica também bloqueado, à espera de uma invocação. Em todo este processo, o cliente não sabe que a operação foi executada remotamente. Na Figura 2-3 é ilustrado todo este processo; *Call* refere-se à invocação de um procedimento remoto, *Return* à entrega dos resultados da invocação, *Pack* ao empacotamento e *Unpack* ao desempacotamento.

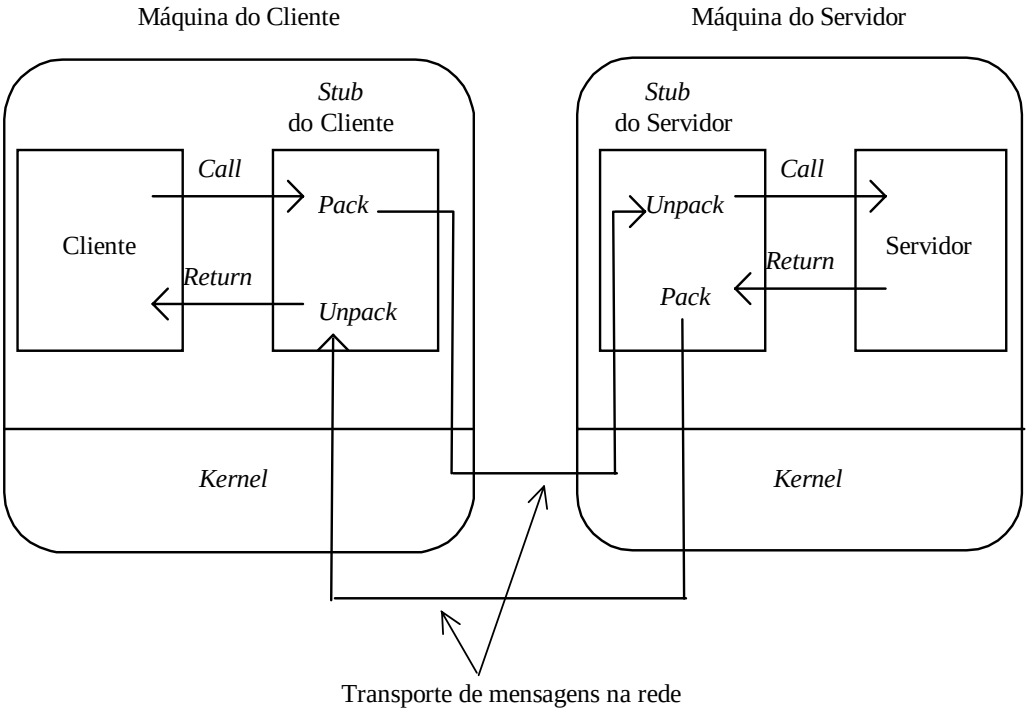

Figura 2-3 – RPC

Uma questão que se levanta é a maneira como o cliente localiza o servidor. Uma hipótese é fornecer ao cliente o endereço de rede do servidor, mas tal seria pouco flexível, pois, se o servidor migrasse, fosse replicado ou a sua interface mudasse, seria necessário encontrar e recompilar numerosos programas.

Para evitar estes problemas, usa-se a ligação dinâmica. Quando o servidor é iniciado, **exporta** (regista) a sua interface para um programa chamado ***binder***, para que a sua existência seja conhecida. Para efectuar o seu registo, o servidor fornece ao *binder* um nome, um número de versão, um identificador e um *handle*, usado para o localizar. Quando o cliente invoca uma operação remota pela primeira vez, o *stub* do cliente envia uma mensagem ao *binder*, pedindo para **importar** a interface do servidor. Se não houver nenhum servidor do tipo pedido pelo *stub* do cliente, a operação falha. Se houver algum servidor do tipo pedido, o *binder* fornece ao *stub* do cliente o *handle* e o identificador do servidor.

Este método de exportação e importação de interfaces é bastante flexível, porque pode, por exemplo, lidar com vários servidores que suportam a mesma interface. Uma

desvantagem que este método apresenta é o *overhead* introduzido pelos processos de importação e exportação de interfaces.

2.5.2. Modelo de três camadas (*3-tier*)

Uma variação ao modelo cliente/servidor é o modelo de três camadas (3-*tier*), no qual é acrescentada uma terceira camada às já conhecidas camadas dos clientes e dos servidores. Esta terceira camada comporta serviços que já existiam antes do desenvolvimento da aplicação distribuída actual e que os servidores utilizam para completar as tarefas que lhe são solicitadas pelos clientes. O servidor encapsula e abstrai totalmente as aplicações legadas, pelo que os clientes nunca interagem directamente com a terceira camada, continuam a interagir somente com os servidores da segunda camada. Um serviço da terceira camada está localizado habitualmente na mesma máquina em que se encontra localizado o servidor da segunda camada que o utiliza. O modelo de três camadas encontra-se esquematizado na Figura 2-4.

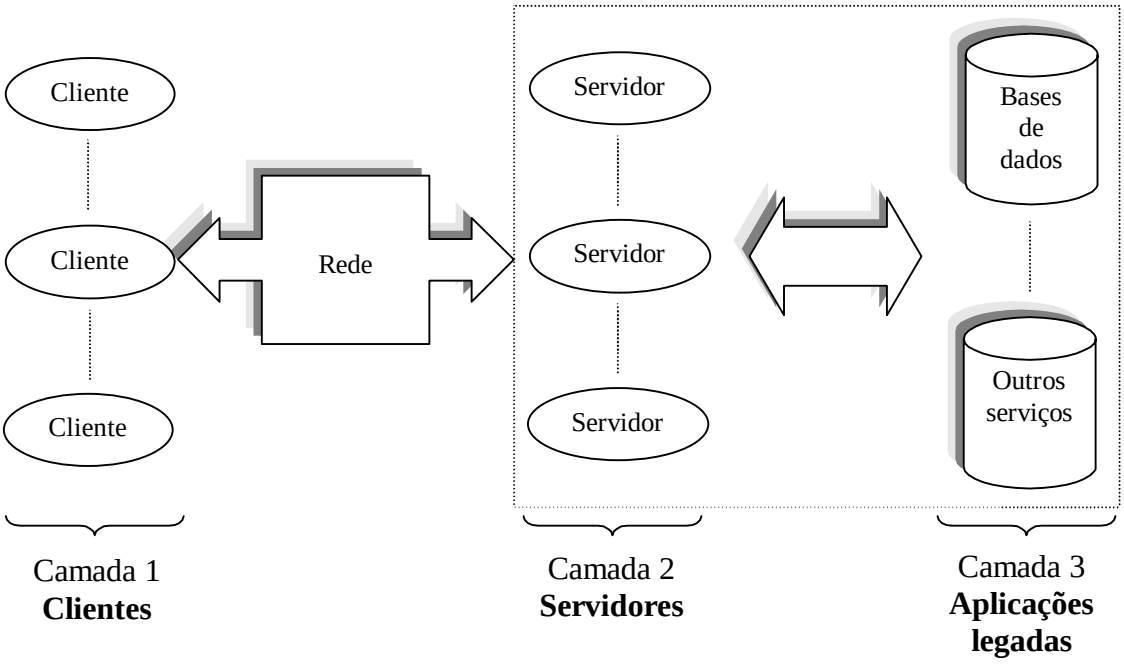

Figura 2-4 – Modelo de 3 camadas

2.5.3. Modelo Cliente-Mestre-Escravo

Uma outra variante do modelo cliente-servidor, usada em aplicações de computação distribuída, é o modelo cliente-mestre-escravo (*client-master-slave*) [5], que desdobra o lado do servidor numa componente mestre e diversos escravos.

Nesta arquitectura, um cliente pede um determinado serviço, eventualmente complexo e envolvendo grande volume de dados, ao mestre, o qual o subdivide e distribui por diversos escravos, permitindo assim, de uma forma paralela, processar grandes quantidades de informação de uma forma mais rápida.

Um exemplo bastante popular da aplicação desta arquitectura é o projecto SETI (*Search for ExtraTerritorial Intelligence*) [6], em que participantes voluntários instalam no seu computador um programa escravo que detecta períodos de inactividade do CPU e aproveita-os para processar grandes volumes de dados obtidos a partir de satélites geostacionários, enviados pelo programa mestre.

2.5.4. Outros modelos

O modelo cliente-servidor é reconhecidamente o modelo de comunicação mais importante, mas existem outros modelos menos utilizados, mas com aplicações específicas que merecem a sua referência.

Um desses modelos é o modelo *Producer-Consumer*, no qual um processo (*Producer*) produz dados que serão enviados a outro processo (*Consumer*) que os irá processar de alguma forma. Um exemplo deste modelo é o caso das impressoras partilhadas, consumidoras dos dados que são produzidos pelas aplicações que a utilizam, para serem imprimidos [7] [8].

Outro modelo também relevante é o modelo *Publishe-Subscribe*, em que um determinado processo (*Publisher*) publicita a possibilidade de produzir dados (publica eventos) e outros processos (*Subscriber*) manifestam o interesse em receber esses dados (subscrevem eventos), quando estiverem disponíveis [9] [10]. Sempre que sejam produzidos dados pelo *Publisher*, este encarregar-se-á de os enviar a todos os processos que os subscreveram. Este mecanismo proporciona comunicação do tipo

"muitos-para-muitos" (*many-to-many communication*) e fornece anonimato aos intervenientes. Um exemplo de serviço que utiliza este modelo é o serviço de "news", no qual os subscritores manifestam o interesse em receber informação de um determinado tópico ou conjunto de tópicos.

2.6. COM/DCOM

A arquitectura **DCOM** (*Distributed Component Object Model*) [11] surgiu como uma extensão do modelo de objectos **COM** (*Component Object Model*), de forma a permitir que componentes situadas em computadores distintos possam comunicar entre si. O modelo COM surgiu associado à tecnologia **OLE** (*Object Linking and Embedding*), que permite integrar diferentes aplicações e tipos de dados num mesmo ambiente aplicativo (e.g., a visualização de uma componente multimédia num editor de texto). Contudo, esta tecnologia possuía inicialmente diversas limitações, as quais foram em grande parte solucionadas na 2ª versão, que recorre ao COM como tecnologia de encapsulamento de objectos.

A DCOM, tal como outras arquitecturas de processamento distribuído, dispõe de uma linguagem de definição de interfaces, fazendo assim uma separação entre a implementação e a respectiva interface, o que facilita a reutilização de componentes. Contudo, não suporta a herança na definição de interfaces, apesar de um objecto poder possuir várias interfaces.

Uma classe DCOM implementa um determinado número de interfaces. Os objectos DCOM são identificados através de um identificador global único (GUID – *Global Unique Identifier*), o qual faz corresponder uma classe a uma DLL (*Dynamic Link Library*) [12] ou a um executável (EXE). Uma DLL é um ficheiro que contém uma ou mais funções que efectuam um determinado trabalho e que podem ser usadas por outros programas, sendo carregadas em memória apenas aquando da sua execução. O registo de todos os identificadores de objectos existentes numa máquina é mantido pelo *system registry* dessa máquina.

Na arquitectura COM/DCOM, quando um cliente faz uma invocação de um método num determinado objecto, se esse objecto estiver localizado na mesma máquina,

o COM intercepta a invocação e entrega-a ao processo do destinatário; caso o destinatário esteja localizado numa máquina remota, é a DCOM que se encarrega da entrega da invocação, usando um protocolo de rede.

A Figura 2-5 mostra as diferentes componentes envolvidas na arquitectura DCOM. A componente COM *runtime* fornece serviços orientados a objectos às componentes cliente e servidor e usa os mecanismos de segurança e o DCE RPC para gerar pacotes de rede que possam ser usados pelo DCOM. Note-se ainda que uma componente DCOM pode-se comportar como cliente e como servidor.

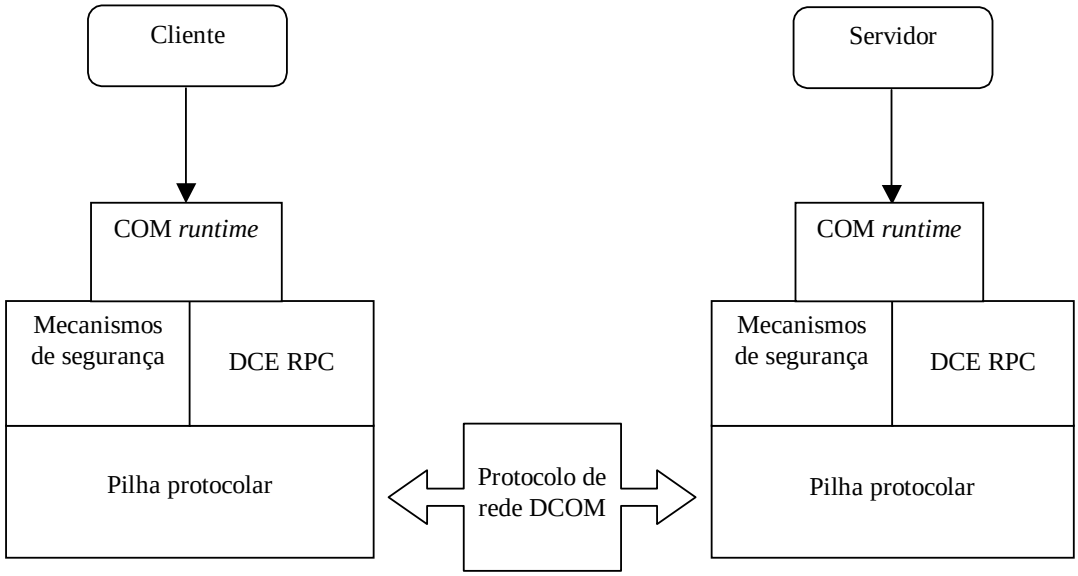

Figura 2-5 – Arquitectura DCOM

No desenvolvimento de aplicações distribuídas é comum surgirem alguns conflitos e/ou restrições:

- componentes que interajam mais frequentemente devem estar localizadas o mais próximo possível, de forma a minimizar a latência das invocações;
- algumas componentes devem estar localizadas numa máquina ou num local específico;
- a opção por componentes pequenos aumenta a flexibilidade, mas também aumenta o tráfego na rede. Por outro lado, componentes maiores diminuem o tráfego, mas também a flexibilidade.

Na arquitectura COM/DCOM, é a mesma a forma como um cliente invoca uma componente no mesmo processo, em diferentes processos da mesma máquina ou numa máquina remota. De facto, não existem quaisquer diferenças no código-fonte, nem sequer é necessário efectuar qualquer recompilação. Sendo uma extensão do COM, a DCOM é independente da linguagem de programação.

Numa aplicação distribuída, o servidor necessita de ser notificado acerca da desactivação dos seus clientes. Este aspecto é particularmente importante no caso de existirem falhas de rede ou de equipamentos. Para o efeito, a DCOM mantém um contador para cada componente, de forma a poder controlar o número de clientes que lhe estão ligados. O controlo de actividade dos clientes é efectuado recorrendo a um protocolo de *ping*, assumindo-se o cliente desligado quando se recebem 3 *timeouts* como resultado de um *ping* a esse cliente.

Um aspecto importante na construção de aplicações distribuídas é a escalabilidade. A arquitectura DCOM procura equilibrar o número de *threads*, de forma a nem sobrecarregar a gestão de informação contextual devido à existência de demasiados *threads*, nem provocar congestão dos processadores por existirem poucos *threads* de grande dimensão. Outra forma de incrementar a escalabilidade dos sistemas é através da distribuição das componentes por diversas máquinas.

Outro aspecto que está também relacionado com a escalabilidade é o controlo de versões. Na arquitectura DCOM, se for adicionada funcionalidade a uma componente, a nova versão continua a suportar os clientes da versão antiga. Da mesma forma, os clientes da nova versão podem interagir com servidores da versão antiga.

O tempo necessário para completar uma invocação remota depende do tamanho dos parâmetros da invocação, da distância entre as componentes envolvidas e do número e tipo de elementos activos existentes no caminho percorrido pelos pacotes na rede. O DCOM procura minimizar estes *overheads* recorrendo a algumas optimizações possibilitadas pelo protocolo UDP. Uma dessas optimizações consiste em agrupar as chamadas destinadas a uma determinada componente – por exemplo, se um cliente necessita de fazer 100 invocações a uma determinada componente, o objecto *proxy* dessa componente existente do lado do cliente agrupa essas invocações numa só ou em

blocos. Outra optimização consiste no agrupamento dos *pings* destinados aos diversos clientes de uma máquina.

Os mecanismos de segurança do DCOM são fornecidos pela configuração para cada componente de uma lista de controlo de acesso, semelhante ao que é efectuado no Windows NT pelos administradores de sistema para controlar o acesso a ficheiros e pastas.

As falhas de rede e do lado do cliente são detectadas através do mecanismo de *ping* já referido anteriormente. Quando a falha ocorre do lado do servidor, uma componente intermediária procura restabelecer a comunicação com esse mesmo servidor ou procurando um outro com a mesma interface.

A arquitectura DCOM proclama a completa independência da linguagem de programação, o que é conseguido através da utilização de uma norma binária (COM), a qual pode ser entendida por qualquer linguagem que a suporte. Naturalmente, isto tem algumas limitações, pois não existem mapeamentos normalizados para nenhuma linguagem e o número de linguagens que suportam a DCOM é ainda reduzido.

2.6.1. COM+

Recentemente, com o lançamento do sistema operativo Windows 2000, surgiu um conjunto de serviços complementares à arquitectura COM/DCOM, designado COM+ [13] [14] [15]. Os serviços disponibilizados pelo COM+ procuram suprimir algumas lacunas da arquitectura COM/DCOM, merecendo especial destaque os seguintes aspectos:

- possibilidade de associação de metadados a objectos;
- *Interceptors* – objectos que inter mediam invocações entre cliente e servidor – semelhantes aos *stubs*;
- Serviço de Eventos;
- *Queued Components* – desenvolvimento de aplicações assíncronas e aplicações *offline* – quando um servidor não está disponível, guarda a invocação e executa-a mais tarde;

- *Object Pooling* – ao invés de criar e destruir objectos consoante as necessidades, cria um conjunto de instâncias prontas a usar – do ponto de vista da gestão de recursos é uma solução discutível;
- *Load Balancing* – possibilidade de reencaminhamento de pedidos para máquinas com utilização mais reduzida;
- suporte de XML;
- facilidades de gestão;
- diversos melhoramentos nos serviços de segurança, concorrência, sincronização e *threading*.

2.7. Java/RMI

A linguagem de programação Java surgiu nos últimos anos com o intuito de permitir o desenvolvimento de aplicações capazes de executar em qualquer plataforma que suporte uma máquina virtual Java (**JVM** – *Java Virtual Machine*). Esta capacidade é particularmente útil no desenvolvimento de aplicações para a Internet, através da utilização de ***applets***, pequenos componentes Java que são carregados para um *browser* durante uma consulta a uma página HTML (*HyperText Markup Language*).

O meio básico de comunicação entre JVMs no Java é o *socket*. Contudo, estes exigem à aplicação a codificação e descodificação explícitas da informação trocada. Numa primeira aproximação, a chamada de procedimentos remotos, RPC (*Remote Procedure Call*), seria uma alternativa aos *sockets*, dando a ilusão de estarmos sempre a efectuar invocações locais, sem nos preocuparmos com a codificação e descodificação dos parâmetros e valores de retorno. Essa codificação é feita usando uma representação de dados de rede, o XDR (*eXternal Data Representation*). Contudo, esta aproximação procedimental não satisfaz a comunicação entre objectos.

A RMI (*Remote Method Invocation*) [16] é um *framework* do Java (que já se encontra incluído nas versões 1.1 e seguintes do JDK) que permite a um objecto que se encontre a executar numa JVM invocar métodos de outro objecto noutra JVM. Contudo, a sintaxe de invocação de um objecto remoto é a mesma da de um objecto local.

Tal como noutras arquitecturas distribuídas, os objectos possuem interfaces para as invocações. Uma classe que pretenda implementar uma interface remota, deve estender algumas classes definidas no pacote `java.rmi`. Para que se possam efectuar as invocações, o objecto invocante precisa de conhecer a referência da interface do objecto que deseja invocar, a qual pode ser obtida no serviço de nomes da RMI, designado **`rmiregistry`**, ou ser passada como parâmetro ou valor de retorno de uma invocação.

Uma característica importante da RMI, designada **serialização de objectos**, é a possibilidade de retornar ou passar como parâmetro de uma invocação um objecto. Esta particularidade possibilita a criação eficaz de **agentes móveis** [17] [18].

Se numa invocação forem passadas como parâmetro duas referências para o mesmo objecto, essas referências dirão respeito a uma única cópia do objecto na JVM remota. A esta propriedade dá-se o nome de **integridade referencial**.

Uma aplicação RMI é composta geralmente por dois programas, o cliente e o servidor. Tipicamente, o servidor cria um determinado número de objectos, torna acessíveis as suas referências, publicando-as no `rmiregistry`, e aguarda invocações de clientes. Por sua vez, o cliente adquire uma ou mais referências de objectos remotos e invoca os seus métodos. A RMI fornece os meios que possibilitam a troca de informação entre as duas componentes, de forma a criar a ilusão de se tratar de invocações locais; possibilita ainda o carregamento do código das classes, quando se efectua serialização de objectos.

Quando o cliente invoca um método no servidor, não o está a fazer directamente, mas sim num objecto representante do servidor (uma *proxy*), designado ***stub***, que a RMI coloca que lado do cliente. O *stub* é responsável por transformar a invocação local do cliente numa invocação remota ao servidor: inicia a conexão com a JVM remota, faz o empacotamento dos parâmetros da invocação (*marshalling*), aguarda os resultados da invocação remota, faz o respectivo desempacotamento (*unmarshalling*) e entrega finalmente os resultados ao cliente. Do lado do servidor, um outro objecto intermediário, o ***skeleton***, tem uma funcionalidade complementar: aguarda invocações de clientes, desempacota os parâmetros, aguarda que o servidor lhe entregue os resultados, empacota-os e entrega-os à rede, para serem transferidos para a JVM do cliente. Os stubs e os skeletons são gerados usando um compilador apropriado, o `rmic`. Na versão 2 do Java ,o *skeleton* não é necessário.

A arquitectura RMI encontra-se esquematizada na Figura 2-6.

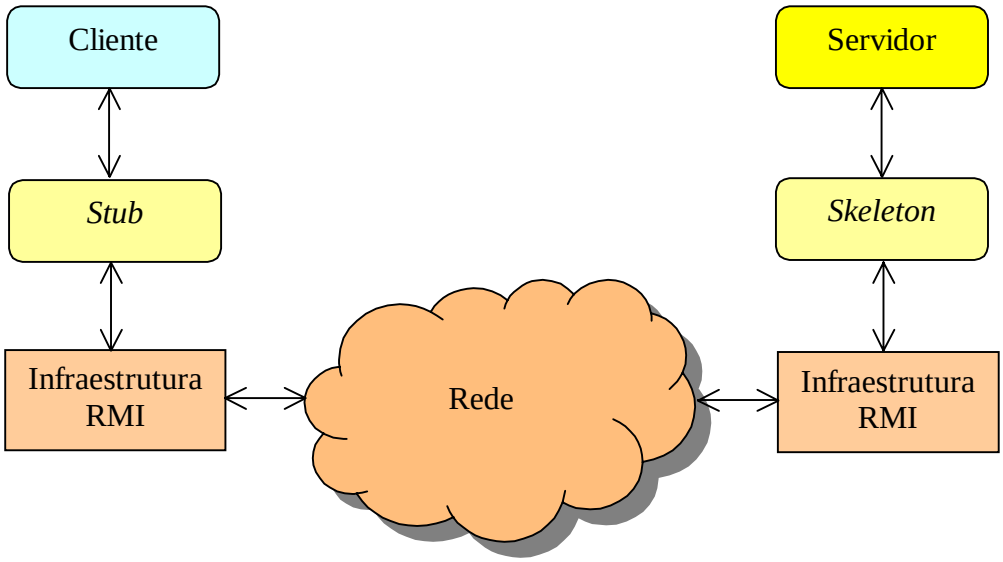

Figura 2-6 – Arquitectura RMI

Na serialização de objectos, quando se desempacotam os parâmetros da invocação, para que estes se transformem em objectos activos, são necessárias as definições das classes relacionadas com esses objectos. Se as classes necessárias não estiverem presentes na JVM local, efectua-se o carregamento dinâmico dessas classes (***Dynamic Class Loading***) a partir da JVM do invocador.

Num sistema distribuído é desejável a existência de um mecanismo que destrua automaticamente os objectos remotos que já não estão a ser invocados por nenhum cliente. A RMI mantém o registo de todas as invocações activas em cada JVM; quando surge uma nova invocação activa numa JVM, o contador de referências do objecto invocado é incrementado e quando essa invocação terminar, o contador é decrementado; quando esse contador atingir zero, significa que o objecto já não está a ser referenciado e é destruído. A este mecanismo dá-se o nome de ***Garbage Collection***.

A RMI beneficia ainda de alguns serviços oferecidos por *frameworks* Java, como o **JNI** (*Java Native Interface*) [19], que permite a integração de sistemas legados, ou o **JDBC** (*Java DataBase Connectivity*) [20], para ligação a bases de dados.

2.8. CORBA

A arquitectura CORBA (*Common Object Request Broker Architecture*) [21] [22] [23] é uma arquitectura de processamento distribuído definida pelo consórcio **OMG** (*Object Management Group*), constituído por diversas empresas e instituições de investigação, com o intuito de permitir a comunicação transparente entre objectos de diferentes plataformas (diferentes tipos de máquinas e de sistemas operativos, modelos de objectos distintos, etc.). A entidade que intermedeia a comunicação é o **ORB** (*Object Request Broker*), fornecendo mecanismos que procuram tornar essa comunicação o mais transparente possível.

Um objecto CORBA é uma componente binária à qual clientes remotos podem aceder invocando os seus métodos. A forma de invocar esses métodos encontra-se descrita em **interfaces**, definidas utilizando uma linguagem apropriada, **IDL** (*Interface Definition Language*), onde se especificam os nomes dos métodos disponibilizados, bem como os seus atributos e parâmetros de entrada e saída. Esta **separação entre interface e implementação** fornece independência em relação à linguagem de programação, possibilitando o acesso a objectos por clientes escritos noutras linguagens. De facto, estão definidos mapeamentos de IDL para diversas linguagens de programação, como é o caso do C, C++, Ada, Java, COBOL e SmallTalk. Outra vantagem inerente a esta característica é a possibilidade de integração de sistemas legados.

A compilação das interfaces descritas em IDL gera um conjunto de objectos que vão ser usados nas invocações estáticas, ***stubs*** do lado do cliente e ***skeletons*** do lado do servidor – o ORB do lado do servidor usa o *skeleton* para invocar o objecto pretendido. Os *stubs* são *proxys* locais para os objectos do servidor, tendo nos *skeletons* os seus interlocutores directos. Entre as tarefas destes objectos encontram-se, por exemplo, a conversão de formatos de representação de dados máquina-rede e rede-máquina (*marshalling* e *unmarshalling*).

Uma característica importante do CORBA é a possibilidade de optar entre dois mecanismos de invocação diferentes (o termo *Common* da sigla CORBA refere-se precisamente a esta característica):

- **estática** – o cliente já conhece *a priori* a interface do servidor;
- **dinâmica** – o cliente descobre a interface durante a execução, após consultar um serviço do ORB que guarda essa informação, designado **interface de invocação dinâmica** (**DII** – *Dynamic Invocation Interface*).

De uma maneira geral, as invocações estáticas são mais fáceis de programar, mais auto-descritivas e mais rápidas do que as dinâmicas. Contudo, estas últimas fornecem um maior grau de flexibilidade. Assim, a invocação estática deve-se usar quando um cliente invoca frequentemente um servidor cuja interface é estável (não muda) e nos serviços que necessitem de rapidez de execução. Quando o serviço não tem requisitos de rapidez, quando o cliente invoca raramente o servidor, ou quando nem sequer o conhece *a priori*, então é preferível usar a invocação dinâmica.

Cada ORB possui um serviço, designado **repositório de interfaces** (*Interface Repository*), que fornece a descrição de todas as interfaces disponíveis, podendo-se dizer que a CORBA é um **sistema auto-descritivo**. Outro aspecto importante diz respeito à transparência quanto à localização. De facto, um ORB pode estar isolado numa única máquina ou ser interligado com outros ORBs de outras máquinas, sendo a comunicação feita usando o protocolo **IIOP** (*Internet Inter-ORB Protocol*). Este protocolo é basicamente uma versão modificada do TCP, com algumas trocas de mensagens definidas pelo CORBA. O CORBA define também um protocolo, designado **GIOP** (*General Inter-ORB Protocol*), que utiliza qualquer protocolo orientado à conexão, não necessariamente o TCP, para a troca de mensagens entre ORBs. Existe ainda uma especificação para um protocolo destinado a conectar com sistemas DCE, designado **ESIOP** (*Environment Specific Inter-ORB Protocol*).

Para que seja possível a um cliente invocar métodos num objecto, tem que conhecer o seu identificador, designado por **referência de objecto**. A sua atribuição é da responsabilidade do ORB, o qual fornece também mecanismos para a sua obtenção por parte dos clientes.

O suporte de mensagens polimorfas é igualmente uma característica muito importante do CORBA. Quando efectua uma invocação, o ORB não invoca meramente um método, invoca um método de um objecto específico. Assim, o mesmo método pode

produzir diferentes resultados, dependendo do objecto que o disponibiliza. Esta característica de **polimorfismo** também distingue uma invocação CORBA de uma comum invocação de procedimentos remotos (vulgo RPC – *Remote Procedure Call*), pois nesta todas as funções com o mesmo nome têm que possuir a mesma implementação.

2.8.1. Arquitectura

A arquitectura CORBA baseia-se no **modelo cliente-servidor**, podendo cada servidor possuir vários objectos que implementam diferentes interfaces. Basicamente, a CORBA é um sistema ***middleware*** que intermedeia as ligações entre clientes e servidores, possuindo para o efeito um conjunto de interfaces de serviços que facilitam a comunicação entre ambas as partes. Na Figura 2-7 encontram-se esquematizados os principais componentes da arquitectura CORBA, bem como os tipos de interacções que se podem estabelecer entre elas. As abreviaturas InterfRep e ImplRep referem-se ao repositório de interfaces (*Interface Repository*) e ao repositório de implementações (*Implementation Repository*), respectivamente.

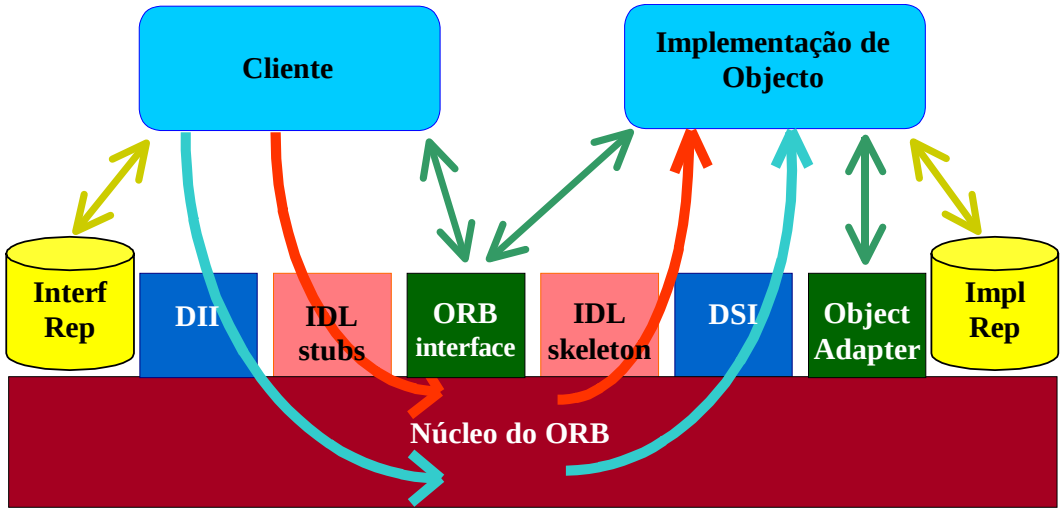

Figura 2-7 – Arquitectura CORBA

A funcionalidade dos *stubs*, dos *skeletons*, do repositório de interfaces e da DII foi já descrita anteriormente. A DII, como já foi referido, permite descobrir em tempo de execução os métodos fornecidos pelos servidores, fornecendo informação relativa às suas interfaces e mecanismos para a geração de parâmetros e recolha de resultados. A **DSI** (***Dymamic Skeleton Interface***) é o equivalente ao DII do lado do servidor.

A **interface ORB** fornece serviços básicos às aplicações como, por exemplo, a conversão de referências de objectos para *strings* e vice-versa. O **adaptador de objectos** (**OA** – *Object Adapter*) fornece os mecanismos para a instanciação de objectos, atribuição das suas referências e multiplexagem dos pedidos e consequente entrega à instância de objecto respectiva. Para além disso, faz também o registo das classes suportadas pelos objectos no repositório de interface. Existem diversos tipos de adaptadores de objectos, destacando-se um adaptador *standard*, o **BOA** (***Basic Object Adapter***), que todos os ORBs devem possuir.

A arquitectura CORBA contempla ainda um **repositório de implementações** (ImplRep – *Implementation Repository*), o qual guarda informações relativas às classes que cada servidor suporta, aos seus objectos instanciáveis e às respectivas referências.

O repositório de interfaces guarda informação relativa às interfaces disponibilizadas por um determinado ORB, a qual é obtida a partir da compilação das interfaces descritas em IDL. Essa informação pode ser usada pelo ORB para verificar a sintaxe das invocações ou para interligar ORB's – as interfaces de objectos dispersos por uma federação de ORB's devem estar definidas em todos os repositórios de interfaces dos ORB's constituintes dessa federação.

2.8.2. Serviços CORBA

Para além dos serviços básicos do ORB, a OMG define um conjunto de serviços CORBA (***CORBA services***) que aumentam e complementam as funcionalidades do ORB. São serviços adicionais, que não fazem parte do conjunto de serviços nucleares de um ORB CORBA, mas que se encontram descritos formalmente na especificação CORBA, sendo comum os fabricantes fornecerem alguns deles junto com o ORB. O número total é de 16, os quais se passam a descrever resumidamente:

- *Life Cycle* – define operações que permitem aos objectos criar, copiar, mover e apagar outros objectos, situados noutros locais;
- *Persistence* – fornece uma interface com funcionalidades para o armazenamento de objectos de uma forma persistente em bases de dados ou ficheiros simples;
- *Naming* – oferece aos clientes a possibilidade de localizarem os objectos através do seu nome (conceito de lista telefónica – páginas brancas);
- *Event* – fornece mecanismos para que os clientes possam ser notificados de eventos específicos ocorridos em determinados objectos;
- *Concurrency Control* – disponibiliza mecanismos para controlar transacções e *threads* concorrentes, através de um gestor de *locks*;
- *Transaction* – fornece mecanismos de suporte a transacções em aplicações que necessitem de acesso concorrente a dados partilhados, de forma a garantir a sua integridade;
- *Relationship* – permite criar associações entre objectos distintos e fornece os mecanismos para manipular essas associações;
- *Externalization* – define mecanismos para guardar a informação de estado de um objecto numa *stream* de dados (exteriorização - *externalization*) e para proceder à sua recuperação para um novo objecto (interiorização - *internalization*) – semelhante a copiar um objecto;
- *Query* – fornece operações de pesquisa em colecções de objectos, assim como operações de inserção, remoção e actualização;
- *Licensing* – dispõe de operações que permitem proceder à tarifação da utilização dos objectos ou controlar quem lhes pode aceder;
- *Properties* – permite associar propriedades aos objectos;
- *Time* – fornece mecanismos para a sincronização temporal em ambientes distribuídos, bem como a definição e gestão de eventos temporais;
- *Security* – suporta operações de controlo de segurança e confidencialidade nas interacções entre objectos distribuídos;

- *Trader* – permite a localização de objectos através do seu nome, com possibilidade de catalogação (conceito de lista telefónica – páginas amarelas);
- *Collection* – fornece mecanismos para o agrupamento de objectos em categorias e para a manipulação dos grupos de objectos (colecções);
- *Notification* – estende o *Event Service*, adicionando-lhe a possibilidade de transmitir eventos usando estruturas de dados bem definidas (e não apenas do tipo *Any*, como acontece no *Event Service*), bem como meios para os clientes refinarem a selecção de eventos num canal.

Na definição dos objectos, através do uso de mecanismos de herança (*inheritance*), podem-se acrescentar as funcionalidades oferecidas pelos diversos serviços CORBA atrás descritos.

2.9. Web Services

O conceito de serviço é muitas vezes associado a uma aplicação acessível através de uma interface, que fornece informação sintáctica acerca da utilização das operações disponibilizadas, como os seus nomes e os tipos de dados que manipula. Das aplicações que seguem este princípio, diz-se, frequentemente, que possuem uma arquitectura orientada a serviços (**SOA** – *Service Oriented Architectures*) [33].

A ***web*** fornece um paradigma de utilização e um conjunto de protocolos de comunicação e de representação de dados de aceitação e suporte generalizados. A possibilidade de construir serviços que sejam acessíveis usando protocolos *web* mostrou-se bastante atractiva para uma parcela considerável da indústria de *software* e da comunidade científica internacional, resultando a convergência das arquitecturas SOA e dos protocolos *web*, sob a alçada do *World Wide Web Consortium* (**W3C**) [34], na tecnologia conhecida por ***Web Services*** [35].

Uma definição simples de *web service*, é a de uma aplicação que é acessível através de uma interface, usando protocolos comuns da Internet (ou protocolos *web*). Tal como acontece com o conceito de componente, um *web service*, representa uma

funcionalidade que pode ser reutilizada sem conhecermos a forma como o serviço é implementado. A grande diferença entre ambos os conceitos reside no facto de os componentes habituais serem acessíveis através de protocolos específicos, enquanto os *web services* são acessíveis através de protocolos comuns da *web* (**HTTP** – *HyperText Transfer Protocol*) [36, 37], sendo a informação representada recorrendo a formatos "universais" de representação de dados (**XML** – *eXtended Markup Language*) [38].

A utilização de protocolos e representações de dados de aceitação generalizada, confere aos *web services* aquela que é uma das características mais apreciadas e desejadas nas aplicações distribuídas e que a maioria das arquitecturas de processamento distribuído não conseguiu alcançar eficazmente: a **interoperabilidade**. De facto, a utilização de protocolos de comunicação da *web* evita a utilização de protocolos proprietários, fornecendo a desejada independência relativamente aos sistemas operativos e fabricantes de *hardware* e *software*. Para além disso, a utilização do XML para a representação dos dados, fornece independência em relação às linguagens de programação, permitindo a troca de informação entre aplicações que, apesar de desenvolvidas em linguagens distintas, possuem mapeamentos dos seus tipos de dados para o XML e vice-versa. A independência fornecida pela utilização do XML pode favorecer ainda a transformação de aplicações legadas em novos serviços acessíveis via *web*, possibilitando a sua interacção com sistemas até então inatingíveis. Esta possibilidade permite rentabilizar os investimentos feitos pelas organizações em sistemas computacionais e de informação e expandir as oportunidades de negócio de uma empresa.

Qualquer tipo de aplicação pode ser oferecida como um *web service* e ficar acessível a partir de qualquer ponto, seja numa Intranet, numa Extranet ou até na Internet, de acordo com as necessidades e políticas de acesso aos serviços de uma organização. O consumidor de um *web service* pode ser um utilizador de um simples *browser*, uma aplicação desenvolvida especificamente para esse fim, ou até um outro *web service*.

Uma escolha adequada da granularidade de um serviço pode permitir a combinação de alguns destes num novo serviço, rentabilizando-se os esforços de desenvolvimento através da reutilização e conferindo maior flexibilidade aos sistemas desenvolvidos, pela possibilidade de recombinação de módulos.

O XML é usado pelos *web services* para descrever as suas interfaces e para a codificação das mensagens trocadas nas invocações. A linguagem *Web Services Description Language* (**WSDL**) [39] define um formato para as descrições XML das interfaces dos serviços. As mensagens que são trocadas nas invocações dos serviços, seguem o protocolo *Simple Object Access Protocol* (**SOAP**) [40], que define o formato das mensagens.

Uma característica igualmente importante dos sistemas de *web services* é a disponibilidade de um serviço que permite o registo dos outros serviços, tornando públicas as suas interfaces, para que um utilizador possa descobrir os serviços e a forma de os utilizar. Este serviço designa-se *Universal Description Discovery and Integration* (**UDDI**) [41] e pode ser usado em outros contextos, para além dos *web services*. Aliás, WSDL, SOAP e UDDI, surgiram isoladamente, mas a junção e amadurecimento dos três componentes produziu o que conhecemos como *web services*.

O SOAP é um protocolo que define o formato dos pacotes e os mecanismos de codificação de dados necessários à troca de mensagens XML entre aplicações. Para além de permitir o funcionamento segundo um modelo de troca de mensagens, o SOAP pode operar igualmente num modelo de chamada de procedimentos remotos. O protocolo SOAP foi desenvolvido inicialmente como um protocolo RPC XML para ambientes Microsoft Windows, mas acabou por evoluir para uma especificação protocolar independente de plataformas computacionais e linguagens, no seio do W3C. A especificação SOAP 1.1 [42] define essencialmente mecanismos para a utilização do protocolo HTTP, mas a especificação 1.2 [43] fornece também a possibilidade de se usar o protocolo *Simple Mail Transfer Protocol* (SMTP) [44]. Uma mensagem SOAP engloba um cabeçalho (*SOAP Header*), com informação acerca do conteúdo da mensagem e um corpo (*SOAP Body*), com a informação a transmitir.

O protocolo SOAP define um esquema de serialização de dados normalizado, derivado do *W3C XML Schema Part 2* [45], mas o utilizador também pode definir o seu próprio esquema de serialização dos dados incluídos nas mensagens a trocar. É possível ainda utilizar o que se designa por *SOAP with attachments* [46], que permite a inclusão de informação não-XML nas mensagens SOAP, como, por exemplo, ficheiros multimédia.

A descrição de um *web service* inserida num ficheiro WSDL contém informação acerca das operações que o serviço disponibiliza, dos tipos de dados que manipula, do formato das mensagens que troca, dos protocolos que suporta e de um ou mais pontos de acesso às suas interfaces. Cada ponto de acesso possui um endereço, denominado *Uniform Resource Identifier* (URI) e cada interface pode possuir vários endereços, respeitantes ao acesso a essa interface recorrendo a vários protocolos.

O serviço UDDI permite o registo dos serviços desenvolvidos pelo programador e a sua eventual categorização, para facilitar a sua descoberta e utilização. O UDDI é igualmente um *web service*, com o qual as outras aplicações comunicam através de mensagens SOAP. Para cada serviço, o registo UDDI guarda informação acerca do seu nome, dos serviços que oferece e do seu ponto de acesso, ou seja, a informação contida na descrição WSDL. Um utilizador que pretenda usar um *web service*, pode procurá-lo no registo UDDI, obtendo a descrição WSDL da sua interface, e a partir desta descrição pode construir a aplicação cliente para comunicar com o serviço, habitualmente designada cliente do serviço. Note-se que o registo UDDI é um registo genérico, o qual pode ser usado não só para manter informação acerca de *web services*, como também de outros tipos de aplicações, se assim se desejar.

A Figura 2-8 mostra a arquitectura genérica de um sistema baseado em *web services* e a sequência de actividades desde o registo de um serviço até à sua utilização por um cliente. Para que o serviço possa ser descoberto pelas aplicações, faz-se, antes de mais, o seu registo no serviço UDDI. Quando um utilizador pretende usar um serviço, procura-o no registo UDDI e este devolve-lhe a respectiva descrição WSDL. Após ter construído as mensagens de invocação das operações disponibilizadas pelo serviço, de acordo com a informação obtida no registo UDDI, o cliente pode interagir com o serviço, trocando com ele as mensagens SOAP adequadas.

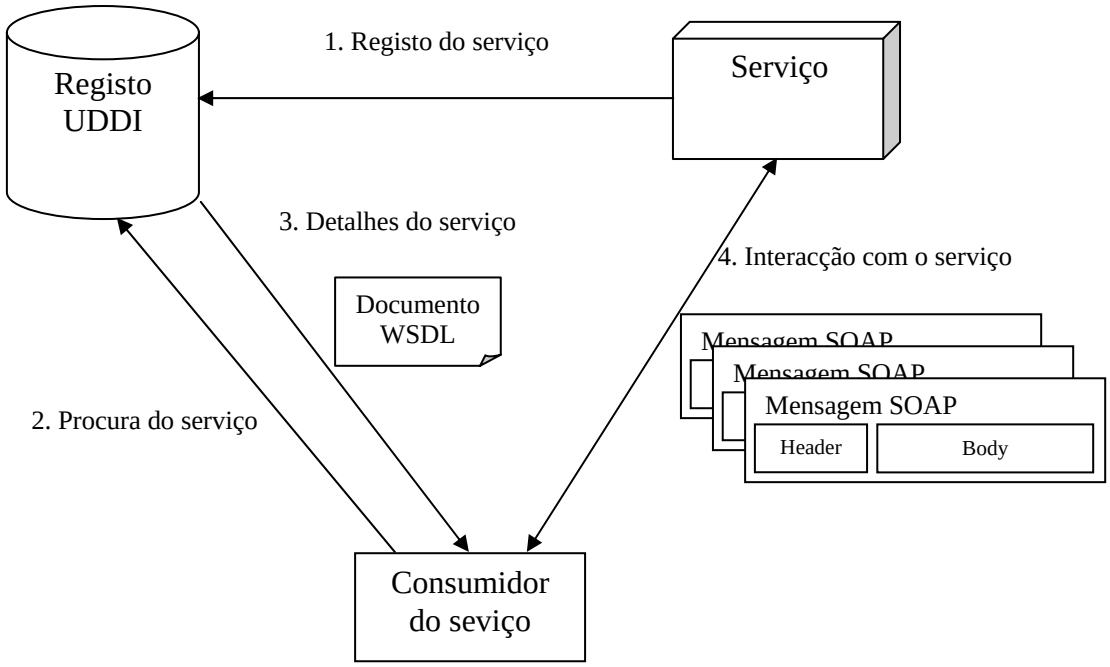

Figura 2-8 – Arquitectura dos sistemas baseados em *web services*

Outra especificação bastante importante no contexto dos *web services* é a especificação *Electronic Business using eXtensible Markup Language* (**ebXML**) [47], que contém uma série de definições específicas aos serviços de comércio electrónico.

As principais características positivas dos *web services* são as que já foram referenciadas anteriormente:

- interoperabilidade em ambientes heterogéneos;
- possibilidade de oferecer serviços comerciais na web;
- integração de sistemas legados;
- grande liberdade de escolha de ferramentas de desenvolvimento, proporcionada pela independência de linguagens e plataformas;
- crescimento do universo de clientes, devido à interoperabilidade;
- produtividade acrescida, quer pelas oportunidades de reutilização, quer pela possibilidade de ir dispondo de sucessivas versões, que, mesmo não sendo as mais completas, permitem ir chegando pelo menos a uma parcela dos potenciais clientes.

No entanto, os *web services* também colocam aos programadores questões para as quais é necessário ter particular atenção:

- relativa juventude da tecnologia faz antever possíveis alterações nas normas entretanto estabelecidas, bem como o aparecimento de novas normas;
- o grau de interoperabilidade já atingido é apreciável, mas não é ainda generalizado, nomeadamente no que diz respeito às linguagens de programação e aos tipos de dados suportados;
- o uso de protocolos abertos de grande disseminação, como o HTTP, coloca problemas de segurança na utilização dos sistemas de informação, principalmente no caso das aplicações empresariais, requerendo especiais cuidados ao nível da autenticação e da troca de informação, sem prejudicar a transparência dos sistemas;
- o uso de *web services* em sistemas de grande escala carece de atenção relativamente aos aspectos de fiabilidade, disponibilidade e escalabilidade, podendo requerer a ponderação de mecanismos de balanceamento de carga ou uma cuidada avaliação da utilização do HTTP como protocolo de comunicação, dada a ausência de garantias ao nível da fiabilidade.

Os *web services* despertaram grande interesse na comunidade científica e na indústria de software, disponibilizando já os principais fabricantes soluções nesta área. Actualmente, os principais produtos de desenvolvimento de *web services* são o Sun ONE [48], o Microsoft .NET Framework [49], o IBM WebSphere [50], o Systinet WASP [51] e o BEA WebLogic [52].

2.10. Comparação entre DCOM, RMI, CORBA e *Web Services*

O objectivo de uma arquitectura de processamento distribuído é o de fornecer um conjunto de mecanismos que possibilitem a comunicação entre processos que executam em máquinas diferentes, de uma forma fiável, segura e o mais transparente possível.

Actualmente, as arquitecturas distribuídas mais divulgadas são a CORBA, a Java RMI, a DCOM e os *Web Services*. A seguir apresenta-se uma comparação dos principais aspectos destas arquitecturas, entre as quais existem alguns pontos que as distinguem e que podem fazer recair a escolha sobre uma delas [53] [54] [55].

Linguagem de programação

A arquitectura RMI está essencialmente presa à linguagem Java; possui uma API, designada JNI (*Java Native Interface*), para tentar suprimir esta falha, mas aumenta substancialmente a complexidade das aplicações, pelo que se revela ineficaz.

O DCOM utiliza uma norma de codificação binária, controlando a forma como cada linguagem é traduzida para o código binário. O resultado desta solução é a existência de grandes diferenças nas traduções das diversas linguagens e mesmo em cada uma delas.

A especificação da arquitectura CORBA define conversores de IDL para as principais linguagens de programação. Da mesma forma, a utilização do XML por parte dos *web services* permite construir mapeamentos para diversas linguagens de programação, o que já acontece para as principais. Esta possibilidade confere às arquitecturas CORBA e de *web services* vantagem no que diz respeito à independência da linguagem de programação, relativamente às arquitecturas DCOM e RMI.

Sistema operativo

Este continua a ser o ponto fraco da arquitectura DCOM, apesar de possuir já suporte para diversas plataformas. Contudo, a sua génese ficou sempre associada às plataformas Windows (principalmente Windows NT), encontrando-se optimizada para esta família de sistemas operativos. Por seu lado, a arquitectura CORBA suporta actualmente um número elevado de plataformas, suportando até mais sistemas Microsoft do que o próprio DCOM. A arquitectura RMI suporta qualquer plataforma que possua uma JVM, o que acontece com os principais sistemas operativos.

Finalmente, os *web services* conseguem praticamente uma total independência do sistema operativo, pela utilização do HTTP e do XML.

Verifica-se, assim, que relativamente à independência do sistema operativo há uma ligeira vantagem dos *web services* relativamente às outras arquitecturas, sendo essa vantagem superior relativamente à arquitectura DCOM.

Comunicação

As arquitecturas CORBA e DCOM usam protocolos que se baseiam no protocolo TCP, IIOP e ORPC (*Object Remote Procedure Call* – baseado no DCE RPC), respectivamente, mas a RMI utiliza um protocolo proprietário, o RMP. No entretanto, este protocolo irá ser substituído brevemente pelo IIOP, na linha da esperada convergência entre as arquitecturas RMI e CORBA. De qualquer forma, trata-se de protocolos específicos das arquitecturas, apesar de serem muito parecidos. Neste domínio, mais uma vez, levam vantagem os *web services*, ao oferecerem mecanismos de comunicação suportados por protocolo *standard*.

Serviços genéricos

Este é um aspecto em que a arquitectura CORBA leva vantagem nítida sobre todas as suas competidoras, definindo 15 serviços genéricos. Por sua vez, a arquitectura DCOM possui apenas implementações limitadas dos serviços de nomes, de transacções e de segurança. No que diz respeito à RMI, implementa apenas os serviços de nomes, de directório, de mensagens e de transacções. Nem DCOM nem RMI implementam serviços igualmente importantes, como é o caso do *Trader*, ou mesmo do serviço de eventos, que apenas surge mais recentemente no COM+. Quanto aos *web services*, a sua relativa juventude faz com que não exista grande trabalho feito nesta área, estando a especificação restringida aos serviços básicos, como o UDDI.

Mensagens assíncronas

As arquitecturas CORBA, RMI e *web services* possuem esta possibilidade. A arquitectura COM/DCOM não a possui originalmente, mas disponibiliza-a com o COM+.

Interfaces dos serviços

As quatro arquitecturas fazem a separação entre as interfaces e as implementações respectivas. No caso da RMI, as interfaces são definidas em Java, não possuindo qualquer linguagem de definição de interfaces, o que prende a RMI à linguagem Java. As arquitecturas DCOM e CORBA possuem linguagens apropriadas para a definição de interfaces, denominadas IDL em ambos os casos. Contudo, a especificação de uma interface em CORBA-IDL é mais simples do que em DCOM-IDL. No que diz respeito aos *web services*, as interfaces são descritas em XML, sendo a sua leitura menos intuitiva do que a do IDL.

Mobilidade de objectos

A arquitectura RMI é a única que possibilita a passagem de objectos como parâmetros de invocações, usando o mecanismo de serialização de objectos, possibilitando a mobilidade de código entre computadores. A especificação da versão 3.0 da arquitectura CORBA contempla já esta possibilidade, não existindo ainda, porém, implementações dessa versão.

Segurança

Qualquer das quatro arquitecturas possui mecanismos de segurança. A arquitectura RMI utiliza o mesmo serviço de segurança da arquitectura CORBA, o qual

é bastante completo, suportando o protocolo SSL (*Secure Socket Layer*) e possibilitando a definição de três níveis de segurança, de acordo com as necessidades de cada ORB. A arquitectura DCOM possui o serviço de segurança do Windows NT, o que, por si só, é já uma grande limitação, independentemente do valor desse serviço, pois acaba por limitar na prática o suporte multi-plataforma. Os *web services* suportam igualmente o protocolo SSL, bem como *XML signature*.

Tolerância a falhas

As arquitecturas DCOM e RMI possuem um mecanismo semelhante. A arquitectura CORBA não contempla na sua especificação nenhum mecanismo de tolerância a falhas, havendo, no entanto, alguns fabricantes que possuem soluções próprias, basicamente semelhantes ao mecanismo usado pela RMI e pelo DCOM.

****

As arquitecturas RMI e DCOM possuem algumas características que as tornam menos adequadas a uma utilização generalizada como suporte a aplicações distribuídas, apesar das evoluções sofridas por ambas.

Das quatro arquitecturas, a que se encontra num estado de desenvolvimento mais avançado é a CORBA, possuindo um conjunto de características e serviços mais completo do que qualquer uma das outras. Nesta vantagem da CORBA, assumem particular importância os aspectos relacionados com as linguagens e plataformas suportadas, com a variedade de serviços disponibilizados e com os mecanismos de segurança. Os desenvolvimentos mais recentes da arquitectura CORBA, nomeadamente a sua versão 3.0 e a RT-CORBA, eliminam algumas das lacunas que a arquitectura possui e tornam-na mais apta para o desenvolvimento de serviços com necessidades de tempo real, como no caso de serviços audiovisuais ou de serviços de controlo de navegação. Contudo, apesar de o CORBA se encontrar em melhor posição do que RMI

e DCOM, no que diz respeito às linguagens e plataformas suportadas, este é um aspecto em que os *web services* mais se destacam. As maiores lacunas do CORBA são a necessidade de possuir um ORB em ambos os lados da interacção (cliente e servidor) e as falhas de interoperabilidade quando esses ORB's são de diferentes fabricantes.

Os *web services* apresentam ainda algumas falhas, por não se encontrarem suficientemente amadurecidos, mas os princípios nos quais se baseiam podem permitir o seu crescimento como infra-estrutura de comunicação distribuída. Uma vantagem que os *web services* apresentam em relação a todas as outras tecnologias é a facilidade de atravessamento de *firewalls*, dada a utilização do porto 80 por parte do HTTP.

De uma forma geral, CORBA e *web services* possuem um conjunto de importantes vantagens relativamente às arquitecturas DCOM e RMI, apresentando ligeiras vantagens e desvantagens entre si. Em termos de áreas de aplicação de uma e de outra tecnologia, a arquitectura CORBA mostra-se vantajosa quando se pretende um modelo de comunicação mais fechado em termos organizacionais, enquanto os *web services* se mostram particularmente adequados às aplicações a disponibilizar na *web*, às aplicações móveis e a aplicações com fortes requisitos de interoperabilidade.

Capítulo 3

Trabalho Cooperativo Suportado por Computador

Neste capítulo faz-se uma descrição geral dos aspectos relacionados com o Trabalho Cooperativo Suportado por Computador. Após uma breve referência histórica, apresenta-se um conjunto de definições de termos associados a esta área. Seguidamente, são descritos os diversos domínios de aplicação, bem como alguns métodos de classificação de sistemas *groupware*. Abordam-se ainda alguns aspectos relacionados com o projecto e a arquitectura deste tipo de sistemas e é feita uma análise das suas vantagens e inconvenientes. Merece igualmente algum destaque o controlo de concorrência a recursos partilhados, descrevem-se as diversas classes de aplicação de sistemas *groupware* e finaliza-se com algumas considerações acerca da utilização da *Web* como suporte à cooperação.

3.1. Introdução

O Trabalho Cooperativo Suportado por Computador, *Computer Supported Cooperative Work* (**CSCW)** [56, 57] na terminologia anglo-saxónica, é uma área científica que surgiu na década de 80, emergindo a partir da Automação de Escritório (*Office Automation*). Esta, por sua vez, apareceu cerca de dez anos antes, como uma forma de suportar o trabalho administrativo de grupos e organizações, evoluindo a partir de sistemas organizacionais como os sistemas de emissão de bilhetes de avião, que tinham surgido na década de 60 [58]. No resto desta tese será usada a sigla CSCW para designar o Trabalho Cooperativo Suportado por Computador, por uma questão de simplicidade e facilidade de identificação da área científica.

De uma forma genérica, o CSCW é uma área científica que estuda a forma como o trabalho de grupo pode ser suportado por tecnologias de informação e comunicação, de forma a melhorar o desempenho do grupo na execução das suas tarefas.

O CSCW enquadra-se num domínio científico interdisciplinar, envolvendo as áreas científicas dos sistemas distribuídos, comunicação multimédia, telecomunicações, ciências da informação e teoria sócio-organizacional. O impacto da utilização deste tipo de serviços nem sempre é positivo, se não se tiverem em consideração factores sócio-profissionais. De facto, convém ter sempre presente que, ao construir aplicações CSCW, poderemos estar a modificar radicalmente práticas de trabalho ou a diluir aspectos organizacionais da equipa, o que pode ter consequências negativas na sua aceitação e adopção. Sempre que possível, devem usar-se metodologias que ajudem a compreender melhor os hábitos de trabalho [59-63]:

- etnografia – observação directa dos hábitos de trabalho;
- projecto compartilhado – envolvimento dos utilizadores no projecto da aplicação;
- pesquisa da acção – observação directa em que se procura ir introduzindo alterações nos hábitos de trabalho, aproximando-os dos que virão a ser seguidos na aplicação.

Desta forma, é desejável e frequente que os projectos de investigação nesta área envolvam pessoas com formações académicas muito diferentes, principalmente nas áreas da Engenharia e da Sociologia, não descurando os próprios destinatários da aplicação.

Os programas informáticos cujo objectivo é serem usados por grupos cooperativos designam-se habitualmente por ***groupware***. Genericamente, pode-se considerar o *groupware* como sendo *software* que suporta CSCW. As aplicações *groupware* mais antigas são o correio electrónico (*e-mail*), os grupos de discussão (*news*) e os sistemas de conversação em modo de texto envolvendo apenas dois interlocutores (*chat*) – sistemas mais recentes, como o *Internet Relay Chat* (IRC) e o ICQ (“*I Seek You”*) [64] já suportam grupos maiores e outros tipos de média. Os sistemas *groupware* enquadram-se na categoria das aplicações distribuídas, nas quais há uma divisão da funcionalidade por um conjunto de módulos cooperantes, dispersos por diferentes computadores.

Os grupos de trabalho em questão não são meros agrupamentos de pessoas, mas sim grupos de pessoas empenhadas na execução de uma tarefa comum ou de um projecto, ou seja, pessoas que formam uma equipa. Essa equipa deve ser constituída por indivíduos cujas competências e/ou talentos se complementem, sendo necessário também que haja uma boa coordenação, de forma a atingir com sucesso os objectivos. Por vezes, a constituição da equipa pode ser diferente consoante as tarefas de um mesmo projecto, sendo este dinamismo facilitado pelo envolvimento de meios informáticos e telemáticos. Para além disso, a constituição dos grupos pode ser intra-departamental, inter-departamental ou mesmo inter-organizacional, pelo que os meios informáticos e telemáticos podem, mais uma vez, ter um papel importante na prossecução das tarefas.

O grau de envolvimento das tecnologias de informação e comunicação no trabalho de grupo também varia, consoante os casos. Assim, se a comunicação de grupo se faz exclusivamente recorrendo ao computador, está-se perante um grupo electrónico. Quando a comunicação pessoal subsiste, apesar da utilização de comunicação por meios informáticos, fala-se em grupos suportados electronicamente.

A interacção entre os elementos de um grupo pode ser mais ou menos frequente, podendo-se classificar em 4 tipos, ilustrados na Figura 3-1. A interacção mais fraca corresponde ao simples acto de informar, enquanto que a interacção mais rica ocorre para o caso da cooperação, em que há uma actividade partilhada para a qual é necessário interagir de uma forma muito frequente. No caso da colaboração, há uma actividade comum, mas a comunicação é esporádica. A coordenação não envolve comunicação com o fim de executar uma actividade comum, mas sim o estabelecimento de contactos com o objectivo de coordenar actividades e informação do grupo.

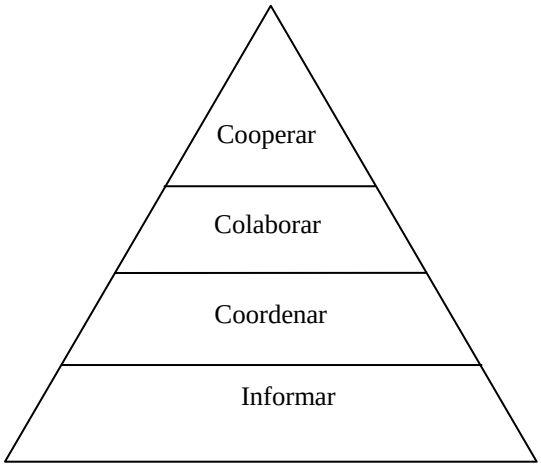

Figura 3-1 – Grau de interacção num grupo

A cooperação em grupos de trabalho pode assumir um carácter conjunto ou um carácter disjunto. No primeiro caso, todos os participantes têm que completar as suas tarefas individuais para que se finalize a tarefa comum. No segundo caso, basta que um dos participantes termine a sua tarefa para que se dê o trabalho por concluído.

Numa equipa de trabalho, a cooperação pode ser utilizada em maior ou menor grau e pode também assumir diversas formas:

- cooperação complementar – cada membro executa uma parcela do trabalho global, o qual se obtém pela junção das várias parcelas;
- cooperação competitiva – vários membros executam a mesma tarefa, como forma de assegurar que alguém a completa ou para seleccionar a melhor solução entre várias (por exemplo, projecto de arquitectura);

- cooperação entre pares (*peer-to-peer*) – cooperação faz-se entre membros com o mesmo tipo de funções na organização;
- cooperação líder-seguidor – um líder de projecto fornece indicações para os restantes membros da equipa seguirem no desenvolvimento do trabalho.

Um aspecto bastante importante para o *groupware* é a existência de um ambiente partilhado pelos vários elementos do grupo. Este ambiente partilhado coexiste normalmente com ambientes privados de cada elemento do grupo, o que implica possuir diferentes mecanismos de gestão de acesso à informação, consoante esta seja privada ou partilhada. Neste último caso, há que ter particular cuidado com os aspectos de controlo de concorrência, de forma a assegurar a consistência da informação partilhada.

Potenciais domínios aplicacionais da utilização de *groupware* são o projecto e desenvolvimento de *software*, a coordenação de processos de trabalho (*Workflow*), o ensino, a telecooperação, a arquitectura, o projecto de engenharia, a telemedicina e a edição cooperativa.

3.2. Classificação de sistemas *groupware*

Como foi referido anteriormente, existem diversas categorias de sistemas *groupware*, os quais podem ser agrupados segundo diversos critérios. Os critérios de classificação de *groupware* mais comuns são a taxionomia tempo-espaço, os domínios de aplicação e o modelo 3C. Existem, contudo, outros modelos de classificação menos utilizados, por serem algo limitativos, quer na quantidade de sistemas *groupware* que conseguem abranger, quer na forma como o seu agrupamento é feito, revelando-se demasiado genérico:

- modelo quantitativo: a classificação é feita de acordo com o número de membros do grupo;
- modelo social: faz-se a distinção consoante a comunicação dentro do grupo é formal ou informal;

- modelo organizacional: distingue as reuniões de grupo entre presenciais e electrónicas.

3.2.1. Taxionomia tempo-espaço

O critério mais utilizado para classificar os sistemas *groupware* é a taxionomia tempo-espaço, na qual se faz uma classificação de acordo com a localização das pessoas e com o momento no qual se dá a interacção. Genericamente, a cooperação pode ocorrer em tempo real ou em instantes distintos e no mesmo local ou em locais geograficamente dispersos.

A classificação segundo a localização dos intervenientes no processo de trabalho distingue os sistemas em que as pessoas estão localizadas no mesmo espaço dos sistemas em que esse espaço é disjunto. Neste último caso, considera-se ainda o facto de a localização das pessoas ser previsível ou imprevisível, ou seja, há casos em que se sabe sempre *a priori* qual é a localização de cada interveniente (inclusive, poderá ser sempre a mesma) e há casos em que a localização de um ou mais intervenientes pode variar frequentemente (utilizadores móveis).

No que diz respeito ao momento no qual se dá a cooperação, este pode ser o mesmo, falando-se de cooperação síncrona (em tempo real), ou pode ocorrer em instantes distintos, ou seja, os intervenientes não actuam simultaneamente. Neste último caso, também se leva em consideração o facto desses instantes serem previsíveis (em maior ou menor grau) ou serem totalmente imprevisíveis.

A Tabela 3-1 mostra as diversas categorias de sistemas, de acordo com este método de classificação.

ESPAÇO \ TEMPO		SIMULTÂNEO (SÍNCRONO)	INSTANTES DIFERENTES (ASSÍNCRONO)	
			PREVISÍVEIS	IMPREVISÍVEIS
MESMO ESPAÇO		Reuniões presenciais	Trabalho por turnos	Quadro
ESPAÇOS DISTINTOS	PREVISÍVEIS	Videoconferência	Correio Electrónico	Edição de documentos
	IMPREVISÍVEIS	Conferência móvel	Conferência sem requisitos de tempo real (e.g. Usenet)	*Workflow*

Tabela 3-1 – Taxionomia tempo-espaço

3.2.2. Domínios de aplicação

Outra forma de classificar sistemas *groupware* é a classificação de acordo com os domínios de aplicação. Este método divide as aplicações *groupware* em sete categorias, de acordo com a sua funcionalidade genérica:

- sistemas de mensagens;
- editores de grupo;
- salas de reunião electrónicas ;
- sistemas de conferência;
- espaços de informação partilhada;
- agentes inteligentes;
- sistemas de coordenação (*Workflow management*).

Nos sistemas de mensagens estão incluídos todos os sistemas de troca de mensagens entre os membros de um determinado grupo, independentemente da interacção ser do tipo um para um, um para muitos ou muitos para muitos, dos tipos de meios que suporta e do momento em que ocorre ou do espaço em que se encontram os intervenientes.

Os editores de grupo são sistemas que permitem a elementos de uma equipa trabalharem sobre um mesmo documento ou ficheiro, de uma forma síncrona ou assíncrona.

As salas de reunião electrónicas são sistemas em que existe uma sala equipada com computadores que fornecem suporte à realização de reuniões de grupos de trabalho. O tipo de suporte fornecido pelos computadores pode ir desde a simples disponibilização de documentos para consulta, até à inclusão de meios mais elaborados, como a edição conjunta de documentos ou os meios de suporte à decisão, incluindo sistemas de votação e recolha de opiniões e ideias.

Os sistemas de conferência permitem a conversação, em tempo real ou não, entre os diversos participantes, podendo utilizar diversos meios de comunicação, desde a simples troca de mensagens de texto (e.g., salas de *chat*), passando pelo correio electrónico, até à utilização de áudio e de vídeo (videoconferência).

Nos espaços de informação partilhada existe uma aplicação que disponibiliza aos elementos de um grupo diversos ficheiros para consulta e alteração. Estes sistemas exigem especial atenção aos mecanismos de acesso à informação partilhada. Nos sistemas mais simples, cada elemento do grupo é responsável por uma parte da informação, não colocando grandes problemas de consistência dos dados. Quando tal não acontece, pode-se efectuar esse controlo recorrendo a exclusão mútua ou ao controlo de versões. Nos casos em que se opta por proporcionar acesso síncrono à informação, com vários intervenientes a trabalhar simultaneamente no mesmo documento, é necessário efectuar o controlo de concorrência, de forma a garantir a consistência da informação.

Os agentes inteligentes são aplicações que procuram substituir os humanos na execução de algumas tarefas, sendo comum encontrá-los, no contexto das aplicações cooperativas, em sistemas de moderação e negociação e em jogos.

Os sistemas de coordenação (*workflow management*) são sistemas que facilitam a coordenação de grupos e de actividades, permitindo que se estabeleça, por exemplo, a ordem pela qual um determinado documento circula pelos elementos do grupo que o irão editar, ou sistemas que permitem a gestão da agenda do grupo em função das diversas agendas pessoais dos seus elementos. Actualmente, muitas grandes empresas e organizações fazem já uso intensivo deste tipo de aplicações.

3.2.3. Modelo 3C

O modelo 3C deve o seu nome ao facto da classificação de sistemas *groupware* incidir sobre o grau de suporte que o sistema fornece a três aspectos básicos: **C**omunicação, **C**oordenação e **C**ooperação. A comunicação diz respeito à troca de informação entre pessoas, a coordenação procura arranjar a melhor maneira de organizar a execução das tarefas por essas pessoas e a cooperação diz respeito ao envolvimento das pessoas na prossecução de objectivos comuns. A Figura 3-2 ilustra este método de classificação de sistemas *groupware*.

Figura 3-2 – Modelo 3C

3.3. Projecto de sistemas groupware

O projecto de sistemas *groupware* é uma tarefa complexa, devido à diversidade de aspectos que lhe estão associados. Para além dos aspectos específicos da aplicação em

si, tem que se considerar os aspectos de cooperação. Ainda antes dos aspectos tecnológicos, assumem particular importância os aspectos psicológicos, sociológicos e de transição de paradigma laboral.

Os aspectos psicológicos têm a ver essencialmente com as consequências introduzidas no trabalho individual de cada membro. Assim, o facto de haver uma maior concentração numa tarefa não implica que ela seja executada mais rapidamente, pois a ausência física dos intervenientes pode dificultar a obtenção de opiniões e consensos. Outro aspecto psicológico importantíssimo é a eventual resistência à mudança, especialmente em comunidades menos predispostas à inovação tecnológica.

Dos aspectos sociológicos, ressalta o efeito da utilização de computadores na estrutura da equipa, podendo dificultar a percepção de hierarquias.

Quando se dá uma **transição** de trabalho individual para trabalho em equipa, há que assegurar que essa transição se faça da forma mais suave possível, sem grandes sobressaltos. Para tal, deve-se tentar introduzir o mínimo de alterações visíveis em relação aos seguintes aspectos:

- meios de comunicação – preservação das capacidades gestuais, por exemplo;
- modos de trabalho;
- faseamento das actividades;
- tecnologia – utilização de tecnologias já habituais, sempre que possível; interface homogénea;
- tempo – historial deve permitir entrada e saída de membros da equipa.

Um aspecto bastante importante para o sucesso de aplicações *groupware* é o benefício que a aplicação pode trazer aos diversos membros da equipa. De facto, diversas aplicações *groupware* têm falhado devido ao facto de trazerem benefícios para uns, enquanto complicam o trabalho de outros, não havendo, assim, uma motivação generalizada que sirva de suporte à sua implantação no seio da organização. Outro factor de insucesso é, por vezes, a violação de tabus sócio-profissionais, como, por

exemplo, o excesso de vigilância que pode constituir um sistema de *workflow* que mantém permanentemente informação acerca do que está a ser feito e por quem.

Trabalhos que habitualmente sejam bastante dependentes da capacidade de improviso dos intervenientes são trabalhos que não beneficiarão da introdução de estruturas de trabalho rígidas.

Na construção da uma aplicação *groupware*, deve-se ter o cuidado de construir uma interface gráfica que não introduza muitas diferenças em relação à sua equivalente *single-user*.

De uma forma geral, deve haver massa crítica suficiente para que seja viável a introdução de *groupware* numa organização. Para além disso, é muito difícil fazer generalizações, dada a grande heterogeneidade de pessoas e situações que estes processos envolvem, pelo que cada caso deve ser cuidadosamente analisado.

No que diz respeito aos aspectos tecnológicos das aplicações *groupware*, assumem particular importância o acesso concorrente dos vários membros a informação partilhada e o suporte à comunicação entre os membros da equipa.

O desenvolvimento de aplicações *groupware*, tal como o de outros tipos de aplicações, é um processo iterativo, no qual não existe uma sequência predefinida de fases de desenvolvimento, mas sim uma sucessão de avanços e recuos de umas fases para as outras. No caso específico das aplicações *groupware*, assume particular importância a observação e compreensão dos métodos de trabalho da organização, permitindo inserir na aplicação cooperativa características que possibilitem uma correcta adaptação das pessoas às novas ferramentas e paradigmas de trabalho. Por exemplo, a comunicação visual é necessária na maioria das aplicações cooperativas, pois as pessoas utilizam frequentemente gestos que podem dar indicações importantes para a cooperação. Em aplicações de suporte a reuniões electrónicas, para além dos documentos finais é igualmente importante guardar documentos intermédios, como esboços e esquemas, que podem mais tarde ser extremamente úteis noutras fases do projecto em que a equipa se encontra envolvida.

Uma questão que se coloca frequentemente no início do projecto de uma aplicação cooperativa é o dilema entre desenvolver toda a aplicação de raiz ou aproveitar sistemas legados, ou seja, arranjar forma de partilhar aplicações já existentes.

Esta última abordagem tem a vantagem de tornar o desenvolvimento da aplicação muito mais rápido e de exigir menor esforço de habituação por parte dos utilizadores. Contudo, pode colocar sérios problemas à sua utilização eficaz, pois, geralmente, não possibilita o controlo de acessos concorrentes à informação.

Outra decisão importante, no decurso do desenvolvimento de aplicações *groupware*, diz respeito à sua arquitectura ser centralizada ou replicada. Numa arquitectura centralizada existe uma aplicação, localizada num servidor, que serve diversos clientes, que apenas possuem uma interface gráfica, conforme se pode ver na Figura 3-3. Este tipo de arquitectura facilita o aproveitamento de sistemas legados. Numa arquitectura replicada há uma repetição de partes ou da totalidade da funcionalidade da aplicação em diversos computadores, devendo um deles assumir o papel de gestor. Este tipo de arquitectura proporciona melhores tempos de resposta, pelo facto de muitas operações serem executadas localmente. Contudo, coloca alguns problemas relacionados com a gestão da aplicação, nomeadamente os que dizem respeito à entrada e saída de membros do grupo, à consistência da informação e ao ordenamento de eventos. A arquitectura replicada encontra-se ilustrada na Figura 3-4.

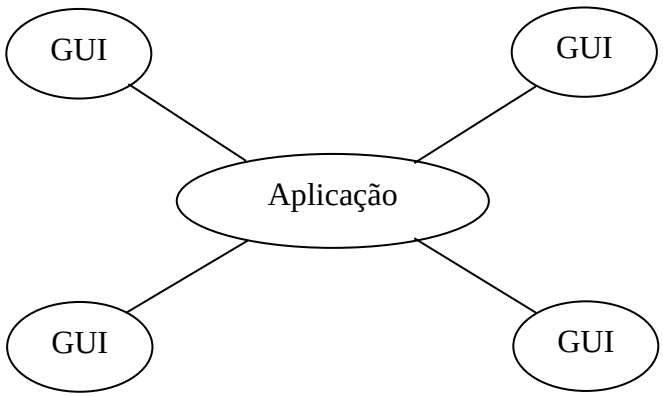

Figura 3-3 – Arquitectura centralizada

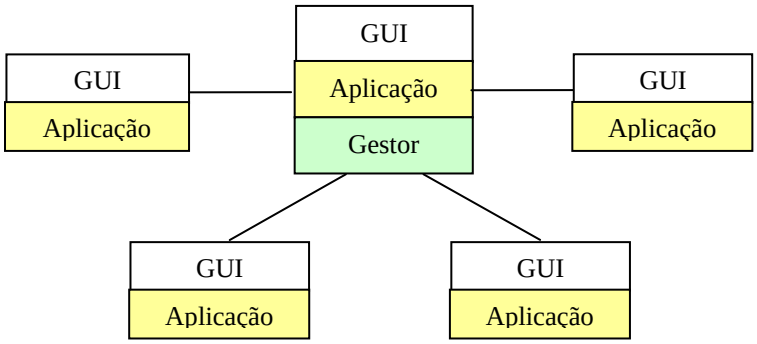

Figura 3-4 – Arquitectura replicada

3.4. Vantagens e desvantagens do *groupware*

Como se tem visto, é grande a quantidade de motivações e problemas que se encontram relacionados com a utilização de sistemas *groupware*. Obviamente, as vantagens e inconvenientes da utilização de sistemas *groupware* podem ser vistos por diversos prismas. A Tabela 3-2 mostra as principais vantagens e desvantagens da utilização de sistemas *groupware*, de acordo com as funcionalidades da aplicação.

Funcionalidade	Vantagens	Desvantagens
Comunicação	Facilidade de interacção dentro do grupo Diminuição de viagens	Custos de instalação Risco de violação da esfera privada
Espaço de trabalho partilhado	Cooperação em tempo real com pessoas dispersas geograficamente	Restrições de interoperabilidade, por falta de normas
Informação partilhada	Partilha de conhecimentos do grupo pode facilitar a coordenação de trabalho	Riscos de segurança contra intrusos
Workflow	Melhoria da coordenação de actividades	Falta de flexibilidade
Suporte electrónico a reuniões presenciais	Melhoria de produtividade, através da disponibilização de informação a todos	Custos de instalação, com risco de não rentabilização do equipamento Treino específico, eventualmente

Tabela 3-2 – Vantagens/desvantagens do *groupware*, na óptica das funcionalidades

Outra forma de ver as vantagens e inconvenientes dos sistemas groupware é de acordo com a posição que uma determinada pessoa ocupa dentro do grupo de trabalho, como se pode ver na Tabela 3-3.

Posição do utilizador	Vantagens	Desvantagens
Gestor	Rapidez de comunicação Acesso a informação relevante Controlo de progresso, em *workflow*	Sentimento de exclusão por pessoas com pior desempenho Redução de encontros presenciais
Perito	Facilidade de comunicação com colegas Aconselhamento facilitado Melhoria de criatividade e inovação (maior interacção)	Perda de tempo em comunicação Risco de falhas em informação sensível, em comunicações com exterior
Secretário	Agendamento Diminuição de papel Diminuição da necessidade de procurar as pessoas	Se as pessoas não utilizarem a aplicação groupware, incrementa-lhe substancialmente o trabalho

Tabela 3-3 – Vantagens/desvantagens do *groupware*, na óptica do utilizador

3.5. Estudo de grupos de trabalho

No projecto de aplicações cooperativas um aspecto bastante importante para o sucesso da aplicação é o estudo prévio do grupo de trabalho ao qual ela se destina. Os resultados desse estudo podem evitar que se cometam determinados erros na concepção da aplicação, os quais podem determinar o seu insucesso. Existem diversos métodos de estudo de grupos de trabalho, os quais se passam a caracterizar resumidamente:

- estudos de campo: observação passiva dos grupos de trabalho. A duração do estudo pode constituir um problema, mas este método possui uma grande autenticidade;
- experiências de campo: simulação de estudos de campo, mas com possibilidade de influenciar alguns processos, como forma de estudar o efeito de algumas alterações na dinâmica do grupo. Diminui a

autenticidade, mas permite ensaiar o efeito de eventuais alterações que sejam introduzidas na futura aplicação cooperativa;

- experiências laboratoriais: criação de um grupo só para a experiência, com a possibilidade de controlar todos os parâmetros externos. Este método é pouco realista e pouco generalista, mas permite determinar padrões comportamentais;
- simulação experimental: estudo laboratorial no qual se tenta reproduzir uma situação real. Sofre dos mesmos problemas das experiências laboratoriais;
- inquéritos: método fácil de implementar e bastante generalista, mas que pode ser pouco fiável, pois algumas pessoas podem não ser suficientemente honestas nas suas respostas;
- teoria formal: método teórico, no qual se modelam grupos e se analisam os seus hábitos de trabalho;
- simulação computorizada: modelação de um sistema real em computador. Possui melhor informação de contexto do que a teoria formal, mas menor capacidade de generalização.

Conforme se verifica pela análise dos vários métodos, todos eles possuem vantagens e desvantagens, pelo que a solução ideal passa pela combinação de vários métodos, obtendo-se assim um conjunto mais completo de directrizes, pelo cruzamento das informações obtidas.

3.6. Processos de grupo

A cooperação numa equipa é geralmente uma alternância repetida entre fases de cooperação assíncrona e síncrona, envolvendo, eventualmente, a constituição de subgrupos de trabalho em algumas dessas fases, para resolver parcelas do problema global. Na situação mais comum começa-se por um processo síncrono – reunião

presencial –, seguindo-se diversas fases assíncronas, intercaladas por fases síncronas (por exemplo, reuniões para fazer o ponto da situação).

Um processo de grupo é a especificação da informação, das actividades e das características de um grupo suportado electronicamente, incluindo o contexto da interacção de grupo. O processo de grupo contempla duas partes, uma estática e outra dinâmica. A parte estática engloba os objectivos, a organização, os protocolos de cooperação e comunicação dentro do grupo e o ambiente de trabalho – equipamento, programas informáticos, disposição da sala, etc. A parte dinâmica comporta os documentos, as actividades, as sessões de uma actividade e o estado corrente do processo.

Basicamente, existem três modelos de processos de grupo:

- modelo centralizado: todas as actividades decorrem no mesmo computador. A Figura 3-5 ilustra este modelo;
- modelo distribuído não replicado: actividades distribuídas por vários computadores, mas sem replicação entre eles. A Figura 3-6 mostra um exemplo deste modelo;
- modelo distribuído replicado: actividades distribuídas por vários computadores, com algumas a serem repetidas em mais do que um computador. Na Figura 3-7 encontra-se esquematizado este modelo.

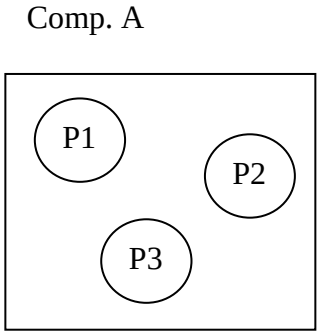

Figura 3-5 – Modelo centralizado

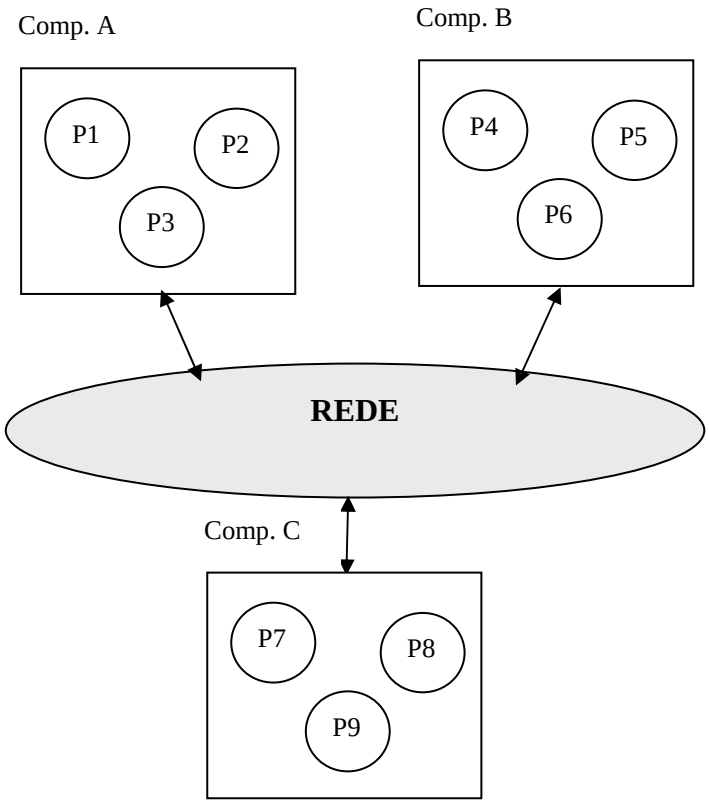

Figura 3-6 – Modelo distribuído não replicado

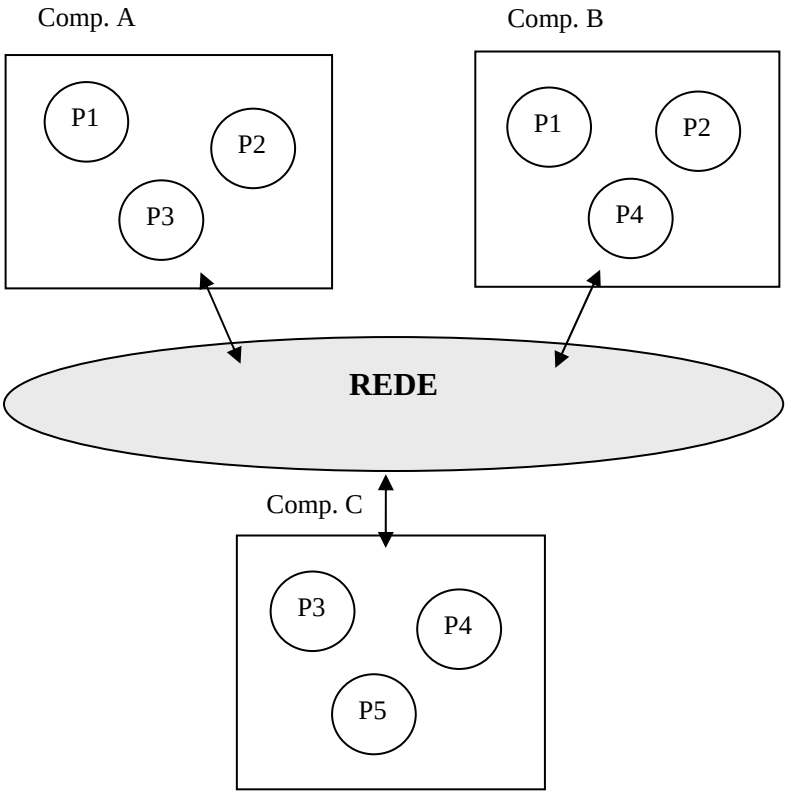

Figura 3-7 – Modelo distribuído replicado

3.7. Comunicação dentro do grupo

Um aspecto importante no funcionamento de grupos de trabalho é a comunicação entre os seus membros, dela dependendo em grande parte o sucesso do grupo na prossecução dos seus objectivos.

A comunicação dentro de um grupo de trabalho pode ser classificada de acordo com diversos critérios:

- segundo a direcção da comunicação – pode ser unidireccional ou bidireccional, consoante a comunicação é feita apenas num sentido ou em ambos;
- classificação temporal – é feita de acordo com o instante em que se efectua a comunicação: na comunicação síncrona ambos os interlocutores estão presentes no mesmo instante, trocando mensagens em tempo real (à semelhança do que acontece numa chamada telefónica); na comunicação assíncrona, uma mensagem é enviada sem se ter conhecimento da presença ou não do interlocutor, o qual a lerá, visualizará ou ouvirá quando for possível (à semelhança do envio de correspondência);
- segundo a distribuição da mensagem:
  - um para um – mensagem com um remetente e um destinatário;
  - um para muitos e muitos para um – mensagem com um remetente e diversos destinatários (difusão) e vice-versa;
  - muitos para muitos – vários remetentes e vários destinatários.
- comunicação directa ou intermediada – consoante as mensagens são trocadas directamente pelos nós ou através de um nó central. A Figura 3-8 ilustra este conceito.

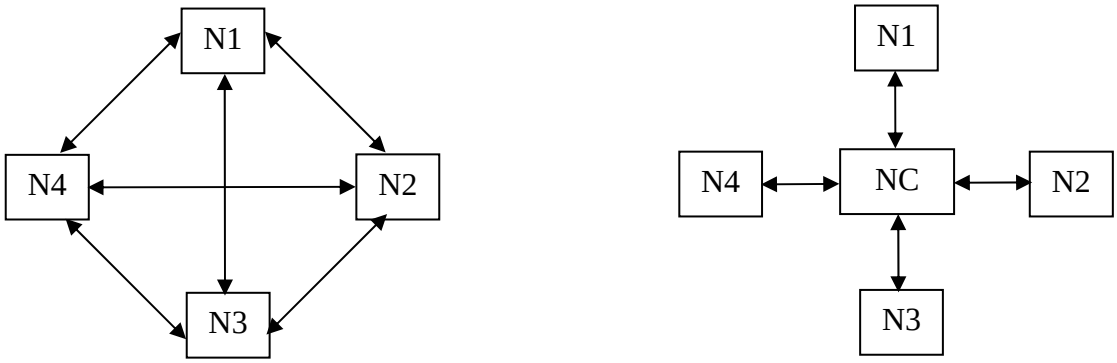

Figura 3-8 – Comunicação Directa *vs* Intermediada

No caso da comunicação assíncrona, existem três modelos de organização das mensagens:

- modelo linear: as mensagens são organizadas cronologicamente, sem qualquer tipo de catalogação, conforme se pode ver na Figura 3-9;

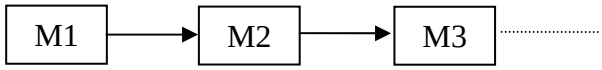

Figura 3-9 – Modelo linear

- modelo combinado: as mensagens são catalogadas por temas, sem que haja relações cruzadas entre os diversos temas. Este modelo está esquematizado na Figura 3-10;

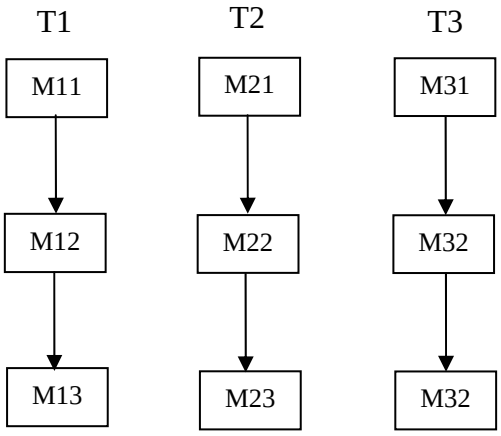

Figura 3-10 – Modelo combinado

- modelo ramificado: neste caso, existe uma catalogação com possibilidade de relacionamentos cruzados entre temas (hiperligações), constituindo ramificações. Este modelo está ilustrado na Figura 3-11.

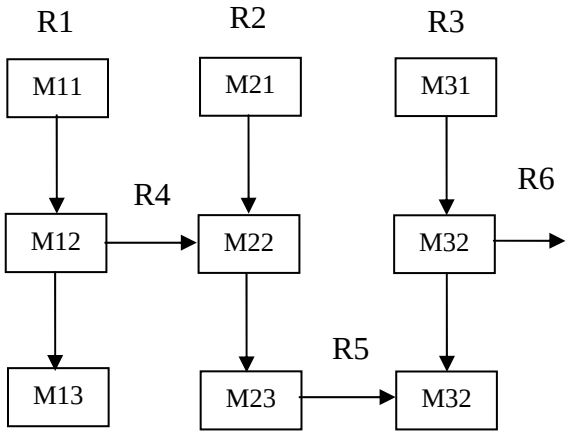

Figura 3-11 – Modelo ramificado

Em determinados grupos de trabalho, pode ser útil possuir uma aplicação que permita ter permanentemente consciência da presença dos outros elementos do grupo, seja através de uma pequena janela de vídeo, seja através de uma representação simbólica do utilizador. Esta capacidade, vulgarmente designada por ***group awareness***, pode incrementar a interacção entre os elementos do grupo.

3.8. Controlo de concorrência

Um dos aspectos tecnológicos mais importantes no domínio das aplicações *groupware* é o controlo do acesso concorrente à informação partilhada, de forma a garantir a sua consistência.

Nos sistemas de gestão de bases de dados, o problema do controlo de concorrência [65, 66] é geralmente resolvido através do suporte de apenas um acesso de escrita em cada instante, ou através de transacções atómicas. Nos sistemas *groupware*, se as aplicações não tiverem grandes restrições de consistência, pode-se resolver o problema através da simples notificação de alterações ou recorrendo a um mecanismo de gestão de versões. Nos casos em que existem restrições mais fortes no que diz respeito à consistência dos dados, tem que se recorrer a protocolos específicos de controlo de concorrência.

Os mecanismos de controlo de concorrência têm que levar em consideração alguns aspectos que podem afectar em maior ou menor grau a sua eficácia e/ou a sua robustez e, consequentemente, a da aplicação subjacente:

- tempos de resposta na visualização das alterações;
- tempos de propagação de notificações;
- tempos de transmissão em redes metropolitanas e de área alargada;
- replicação de dados no computador do utilizador;
- visualização de informação partilhada – conceito *What You See Is What I See* (WYSIWIS);
- robustez após falha;
- mecanismos de notificação de mudanças conflituosas.

A replicação de dados [67] nos computadores dos utilizadores pode ser feita de duas formas: através do envio de toda a informação visual (*display*) ou pelo envio de sinais indicadores das operações efectuadas por cada utilizador.

No paradigma WYSIWIS, todos os utilizadores possuem o mesmo contexto gráfico, ou seja, têm as mesmas janelas abertas e visualizam os mesmos documentos. Contudo, isto pode trazer problemas, pois se todas as acções forem imediatamente reflectidas nos outros utilizadores, pode acontecer que um deles abra uma janela que se vá sobrepor às anteriores, as quais os outros utilizadores poderiam estar interessados em continuar a visualizar. Geralmente, este problema é resolvido pela separação em área pública e área privada, pela possibilidade de personalização da informação visualizada ou pela divergência temporal relativamente ao estado dos objectos partilhados (por exemplo, cada utilizador decide quando deve tornar públicas as suas alterações). Outros elementos habitualmente visíveis nos contextos partilhados são os cursores dos participantes e os teleapontadores. No caso dos cursores, a situação pode-se tornar algo confusa caso existam muitos utilizadores, sendo necessário possibilitar a cada utilizador a escolha entre ver apenas o seu cursor ou os de todos os utilizadores (ou até seleccionar apenas parte deles). O teleapontador simula um vulgar apontador laser e é comum e visível por todos os participantes. A sua manipulação é geralmente atribuída a um utilizador de cada vez, sendo necessário efectuar a sua gestão.

O conceito de contexto partilhado surgiu a partir dos tradicionais quadros existentes nas salas de reuniões, onde é possível fazer pequenas anotações, esboços, etc., como suporte à reunião em curso. Estes quadros possuem algumas limitações físicas, como são o espaço limitado, a necessidade de apagar frequentemente para poder voltar a escrever, perdendo-se o seu conteúdo, ou a eventual ilegibilidade da escrita. Os sistemas *groupware*, quando bem concebidos, podem substituir eficazmente os quadros tradicionais, obviando a estes problemas.

Existem dois tipos de mecanismos de controlo de concorrência: num mecanismo de controlo optimista assume-se que não existirão grandes conflitos no acesso concorrente à informação partilhada, permitindo-se o acesso livremente, não havendo garantia de consistência dos dados em todos os instantes (pode até ser permitido o acesso a dados inconsistentes); num mecanismo de controlo pessimista parte-se do princípio que podem existir graves problemas se não se tomarem medidas adequadas. Desta forma, existem mecanismos de controlo de acesso à informação partilhada, os quais podem ser de dois tipos: centralizado e descentralizado.

No controlo centralizado existe uma entidade que concentra o controlo da concorrência. Essa unidade pode ser uma componente central (unidade de controlo) que serializa e sincroniza todas as operações de escrita, ou pode ser implementada através de um mecanismo de passagem de testemunho (*token passing*), segundo uma ordem predefinida. Neste último caso, cada membro do grupo tem o controlo na sua vez. Esta solução coloca problemas de gestão acrescidos, nomeadamente no que diz respeito à adição e remoção de membros, mas é um protocolo justo, relativamente simples e com garantia de consistência dos dados. A unidade de controlo é também de fácil implementação, mas constitui um ponto de falha único.

O mecanismo de controlo descentralizado distribui a responsabilidade de controlar o acesso aos dados partilhados pelos vários membros. Basicamente, existem mecanismos de controlo descentralizados com e sem votação. Nos mecanismos com votação [68, 69], há um processo de negociação mais ou menos complexo, seguido de votação (com esquemas mais ou menos elaborados), de modo a obter uma decisão coordenada acerca de quem detém controlo sobre os dados partilhados. Os mecanismos de controlo de concorrência sem votação implementam esquemas relativamente complexos para garantir a consistência dos dados (alguns já conhecidos de outras áreas tecnológicas, como os sistemas operativos, por exemplo), dos quais se destacam os seguintes:

- *locks*: bloqueio temporário do recurso partilhado, garantindo o acesso a um único nó em cada instante. A falha deste mecanismo pode implicar o bloqueio permanente de um recurso, exigindo mecanismos para a sua recuperação. O bloqueio pode ser feito a diversos níveis, desde o carácter até ao ficheiro, passando pela linha ou pelo parágrafo. Uma granularidade baixa fornece uma maior liberdade de acção, mas requer um pedido mais frequente de *locks*, resultando num pior desempenho do sistema;
- *floor-passing*: a posse de um recurso vai circulando pelos membros de uma forma não predefinida (na passagem de testemunho há uma ordem predefinida). A passagem de posse do recurso pode ser feita explicitamente entre os utilizadores ou implicitamente, quer através da intervenção de uma unidade central, quer através de negociação entre os utilizadores;

- transacções atómicas [70]: uma operação ou sucede ou falha completamente, sendo mais uma vez importante a granularidade das operações.

Naturalmente, a escolha do mecanismo de controlo de concorrência mais adequado para uma determinada aplicação *groupware* dependerá muito das suas necessidades e da capacidade de suportar a complexidade que essa escolha poderá acrescentar.

3.9. Classes de aplicações *groupware*

O CSCW é uma área científica relativamente recente e, como tal, as aplicações *groupware* são essencialmente experimentais ou aplicações dedicadas, não existindo muitos exemplos de sistemas que tenham conhecido sucesso assinalável. Uma excepção a este panorama é o *Workflow*, actualmente já bastante implantado nas organizações de médio e grande porte. Existem ainda aplicações de comunicação que, pelas suas características, acabaram por ser aproveitadas para formas ligeiras de trabalho cooperativo, como é o caso do correio electrónico, do ICQ ou do NetMeeting [71] (este último, aliás, permite mesmo a partilha de dados e aplicações).

Outro aspecto ainda deficitário do CSCW é a ausência de normalização e de organismos normativos, à excepção, mais uma vez, do *Workflow*, para o qual existe uma organização, o *Workflow Management Coalition* (**WfMC)** [72] a trabalhar na especificação de normas para esse tipo de aplicações.

As aplicações *groupware* podem ser agrupadas de acordo com a sua funcionalidade genérica, constituindo as seguintes classes de aplicação:

- sistemas de comunicação;
- espaços de informação partilhada (*Mediaspaces*);
- coordenação de processos de trabalho (*Workflow*);
- suporte a reuniões (*Workgroup computing*);

- editores de grupo;
- agentes cooperantes;
- ensino assistido por computador;
- realidade virtual.

Nas próximas subsecções descrevem-se resumidamente estas classes de aplicação e indicam-se alguns dos principais sistemas implementados nos últimos anos.

3.9.1. Sistemas de comunicação

A comunicação sempre foi uma capacidade importantíssima para a generalidade das espécies animais, as quais foram desenvolvendo, ao longo do seu processo evolutivo, formas de comunicar mais ou menos elaboradas, algumas delas absolutamente notáveis, como é o caso das abelhas.

O Homem não é excepção a esta necessidade de comunicar e, no decorrer da sua evolução, foi desenvolvendo meios de comunicação cada vez mais diversos e eficazes, desde a fala e a escrita até aos meios telemáticos que hoje usamos em larga escala. O último século foi particularmente profícuo no desenvolvimento de meios de comunicação cada vez mais sofisticados, existindo hoje em dia uma grande dependência dos meios audiovisuais e de telecomunicações, conduzindo mesmo a uma grande alteração do estilo de vida humano e das práticas laborais. No caso do trabalho em equipa, e mais concretamente no que diz respeito ao CSCW, os meios de comunicação assumem um papel primordial, sendo a sua eficaz utilização um importante factor de sucesso.

Correio electrónico

A aplicação de comunicação usando meios informáticos melhor sucedida é o **correio electrónico**. Há mesmo quem considere o correio electrónico como a única

aplicação *groupware* que obteve realmente sucesso. Na realidade, apesar de não ter sido projectado como tal, acabou por se tornar num poderoso meio de suporte ao trabalho em equipa, principalmente pela capacidade de transmitir mensagens textuais, às quais se podem anexar ficheiros dos mais variados tipos. Esta característica permite que circulem documentos entre diversas pessoas, as quais podem efectuar as alterações necessárias ou inserir comentários. Outro factor de sucesso do correio electrónico é a possibilidade de transmitir mensagens para vários destinatários.

Existem, contudo, algumas lacunas que obviam à classificação formal do correio electrónico como uma verdadeira aplicação *groupware*. Um desses problemas está relacionado com o facto de a composição de um grupo de trabalho poder mudar. De facto, o abandono de um elemento do grupo obriga a que existam meios para esse elemento deixar de receber mensagens destinadas a esse grupo. Actualmente, existem diversas aplicações que permitem a criação e gestão de listas de correio electrónico, facilitando algumas tarefas de gestão de comunicação em grupos. Outro problema é o de que um elemento que chega de novo ao grupo poder já ter perdido inúmeras mensagens. Há ainda a questão da organização das mensagens, para a qual o correio electrónico em si não possui capacidades.

Videoconferência

O meio de comunicação preferido em sistemas *groupware* é a videoconferência, pois possibilita o contacto visual entre os participantes, proporcionando um meio de interacção mais rico. Dependendo da utilização que se pretende dar à videoconferência, podemos ter sistemas com qualidade de imagem maior ou menor e com mais ou menos funcionalidades adicionais. Existem basicamente três tipos de sistemas de videoconferência:

- **sala de videoconferência**: sistema de videoconferência em que existem salas especialmente equipadas para esse fim, com diversas câmaras de vídeo e monitores de TV que mostram os participantes remotos a uma audiência presente na sala. Trata-se de sistemas em que a imagem tem uma

boa definição e que se destinam a suportar reuniões ou palestras em grandes organizações, sendo o seu **custo elevado**;

- **videofone**: este sistema é basicamente um telefone que incorpora uma câmara de vídeo e um monitor de TV, ambos de baixa resolução, destinando-se essencialmente a ver a face do interlocutor. O seu sucesso comercial é reduzido, tendo alguma implantação em França e junto do meio jornalístico (situações de indisponibilidade de ligação via satélite para transmissão de sinal televisivo);
- **videoconferência por computador** (***desktop videoconferencing***): neste tipo de sistemas, usam-se câmaras de vídeo de **baixo custo** (habitualmente designadas por *webcams*) ligadas a computadores, sendo a transmissão feita através da Internet. As imagens capturadas pelas câmaras são usadas por programas de videoconferência apropriados, os quais possibilitam frequentemente a utilização de outras funcionalidades – conversação em modo de texto, audioconferência, sistema de mensagens curtas, etc. Existem mesmo sistemas, como o **NetMeeting**, que permitem a partilha de aplicações, proporcionando um meio relativamente simples de cooperação. Este tipo de videoconferência é usado tanto para fins profissionais como para fins lúdicos, devido ao seu baixo custo. A utilização profissional pode colocar alguns problemas de confidencialidade e segurança, pelo que deve ser devidamente ponderada.

A utilização de videoconferência como suporte a grupos de trabalho pode ocorrer essencialmente de duas maneiras:

- como suporte a **reuniões,** em que o estabelecimento da videoconferência é feito esporadicamente, de uma forma planeada ou *ad-hoc*;
- como forma de manter contacto visual permanente com outras pessoas da organização (*people awareness*) – conceito de **telepresença**.

O uso da videoconferência como suporte a reuniões de trabalho pode diminuir a necessidade de deslocações numa organização com instalações dispersas, mas também pode servir de complemento a essas deslocações. Por exemplo, a videoconferência pode ser usada por um elemento de uma organização que se encontra em viagem, para delineamento de estratégias ou para fazer o reporte diário aos seus superiores dos resultados obtidos nos contactos efectuados.

Os sistemas de telepresença [59, 73] possibilitam a visualização de diversas pessoas ou locais de uma organização, de forma a ter-se sempre noção de quem está disponível para interagir ou de quem está presente num determinado compartimento. A visualização directa de pessoas pode ser substituída por algum tipo de sinalética que, por exemplo, indique a sua presença, apesar da indisponibilidade para ser interrompida, como forma de garantir alguma privacidade. A utilização de telepresença associada a locais, como um corredor ou um bar, pode ser um meio de incrementar a interacção informal entre os elementos da organização.

A utilização da videoconferência como suporte a grupos de trabalho possibilita a percepção de gestos e expressões faciais, os quais podem ser semanticamente muito ricos, permitindo extrair informação que não seria perceptível utilizando outros meios de comunicação, como as mensagens escritas ou o telefone. A comunicação visual pode ser particularmente importante em períodos de silêncio, sendo igualmente importante a direcção do olhar ("olhar *olhos nos olhos*") nas situações de comunicação entre duas pessoas. Existe, contudo, alguma dificuldade em assegurar esta última característica no caso dos sistemas *desktop*, visto que as pessoas têm tendência a olhar para o monitor e não para a câmara de vídeo.

A visualização do espaço envolvente dos interlocutores pode ser também desejável, em certas situações, mas implica a disponibilidade de mais câmaras de vídeo ou de imagens de maior dimensão, com um consequente aumento das necessidades de largura de banda e do tamanho da imagem a visualizar. Refira-se, aliás, que o tamanho da imagem pode ser importante para dar uma noção de proximidade do interlocutor [73].

3.9.2. Espaços de informação partilhada

A actividade cooperativa necessita habitualmente de outras funcionalidades para além das de comunicação. Uma funcionalidade bastante importante é a capacidade de partilhar informação entre os elementos cooperantes, permitindo o seu acesso concorrente, quer para leitura, quer para escrita. As aplicações que possibilitam o acesso concorrente a informação, para consulta e alteração, por parte de diversos elementos de um grupo cooperante são habitualmente designadas por Espaços de Informação Partilhada (*Shared Information Spaces* ou ***Mediaspaces***, na literatura anglo-saxónica) [74].

Uma questão importante no contexto dos espaços de informação partilhada é a da organização da informação. Os documentos habitualmente disponíveis em formato de texto, como é o caso dos livros e dos documentos produzidos em processadores de texto tradicionais, são geralmente lineares, ou seja, há uma sequência predefinida de consulta, com a excepção dos dicionários e dos manuais de referência. No caso dos espaços de informação partilhada é preferível dispor de uma organização da informação mais flexível, que permita um acesso rápido a um determinado pedaço de informação. O hipertexto fornece esta não linearidade, através do uso de hiperligações, e fornece ainda a possibilidade de incluir outros tipos de media. Uma desvantagem dos sistemas de hipertexto é a redução de sensibilidade relativamente ao que já se leu e ao que falta ler, facto que se costuma designar por “andar perdido no hiperespaço”, devido em parte a uma má concepção da informação,

O acesso concorrente à informação partilhada coloca problemas de consistência da informação na presença de acessos simultâneos conflituosos, havendo necessidade de fornecer mecanismos adequados que garantam sempre a integridade dos dados. Estes mecanismos de controlo de concorrência foram já descritos na secção 3.8.

Um caso particular de partilha de informação é a procura cooperativa de informação [75]. A sua obtenção pode ser conseguida de diversos modos:

- partilha de resultados de trabalho entre elementos de uma equipa (grupo de pessoas que trabalham em algo comum) ou de uma comunidade (grupo de

pessoas que têm em comum algum aspecto – língua, por exemplo –, podendo nem sequer se conhecerem);

- publicação ou difusão da informação encontrada, por livre iniciativa;
- consultadoria – obtida directamente ou através de sistemas de ajuda à identificação de aconselhamento;
- repositório de grupo, onde é guardada toda a informação que possa ser relevante para os elementos do grupo, a qual pode estar classificada, de forma a facilitar a sua obtenção.

Um exemplo bastante conhecido de sistemas de partilha de informação é o sistema de ajuda colaborativa "Answer Garden 2" [76]. Outro assistente de ajuda colaborativa, o "*Social Web Cockpit*" [77], permite a obtenção de ajuda entre utilizadores de uma comunidade virtual.

O sistema "Agentware" [78] é um sistema de gestão de conhecimento que dispõe de um agente de monitorização que monitoriza constantemente as actividades do utilizador. Se, por exemplo, o utilizador estiver a editar um texto, o agente analisa-o continuamente e gera hiperligações para informação relacionada existente no espaço de informação partilhada.

A Sun® desenvolveu um sistema de suporte a engenheiros que tentam diagnosticar problemas UNIX nos computadores dos clientes, designado "SharedShell Tool" [79]. Este sistema permite aos utilizadores partilharem a linha de comandos e fazer desenhos sobre ela, de forma a conseguirem encontrar da melhor maneira uma solução para o problema.

3.9.3. Coordenação de processos de trabalho

Uma das classes de aplicação de sistemas *groupware* mais difundida é a dos sistemas de coordenação de processos de trabalho, mais conhecidos por sistemas de ***Workflow*** [80, 81]. Este tipo de sistemas permite a especificação e a execução de um conjunto de tarefas, sequenciais ou paralelas, de uma forma coordenada, à semelhança

do que acontece com a produção industrial numa linha de montagem, na qual cada operário é responsável por uma tarefa predefinida, existindo sempre a mesma sequência de tarefas. Exemplos típicos são a atribuição de actividades a elementos de uma organização, a gestão de recursos e a monitorização da evolução das actividades. Refira-se que neste tipo de sistemas não há cooperação em tempo real, mas apenas a coordenação de tarefas de um mesmo processo, mas independentes em termos de execução.

De entre os problemas que a utilização de sistemas *Workflow* pode introduzir destacam-se os seguintes:

- sensação de vigilância electrónica no trabalho;
- baixa flexibilidade, por definição de regras demasiado rígidas;
- necessidade de treino, inerente à adopção de novas formas e ferramentas de trabalho;
- necessidade de homogeneidade da adopção das novas regras de trabalho, sob pena de todo o processo falhar.

Para que os sistemas de *Workflow* possam ser bem sucedidos, devem, dentro do possível, possuir as seguintes características:

- suporte de sistemas legados;
- escalabilidade;
- adaptabilidade;
- possibilidade de execução de tarefas de uma forma sequencial, paralela ou condicional;
- atribuição de tarefas a cargos e não a pessoas específicas.

Para além dos potenciais ganhos em termos de eficiência do trabalho, os sistemas de *Workflow* têm ainda a vantagem de permitir aos novos elementos da organização assimilarem mais facilmente a estrutura dos processos em que se encontram envolvidos. Contudo, a atribuição rígida das actividades pode conduzir, em certos casos, a situações

de forte desmotivação e consequente perda de produtividade, resultantes da limitação de responsabilidade no processo global.

A utilização inter-departamental de sistemas *Workflow* pode não se revelar viável em grandes organizações, visto ser frequente os diversos departamentos utilizarem diferentes metodologias e ferramentas de trabalho, sendo mais comum o seu funcionamento à escala departamental.

São diversas as áreas de aplicação às quais pode trazer vantagens a utilização de sistemas de *Workflow*, mas há um conjunto no qual essa utilização é mais comum:

- escalonamento de processos industriais;
- automação de escritório;
- engenharia de software;
- processamento de documentos;
- reengenharia de processos de negócio;
- centros de assistência a clientes (*Call Centers*);
- CSCW.

Os primeiros sistemas de automação de escritório surgiram na década de 70, mas a sua implantação falhou, em parte devido a aspectos relacionados com a própria actividade de escritório, mas também porque as tecnologias necessárias à sua implementação (computadores e redes) não estavam suficientemente desenvolvidas e difundidas. Somente na década seguinte se tornaram viáveis, principalmente devido ao aparecimento dos computadores pessoais.

Uma parcela importante dos sistemas de *Workflow* é constituída por sistemas desenvolvidos especificamente para uma determinada área de aplicação. Por exemplo, o sistema MILOS [82] é um sistema de *Workflow* que permite a coordenação dinâmica de 1 equipa de programadores. Contudo, existem também alguns sistemas suficientemente genéricos que permitem a sua utilização em diversas situações. Os sistemas genéricos de *Workflow* mais populares são o Lotus Notes (IBM) [83] e o Microsoft Exchange [84].

Podem-se classificar os sistemas de *Workflow* de acordo com a sua arquitectura:

- baseados em correio electrónico: a sequência da execução das tarefas é implementada através da troca de mensagens de correio electrónico e a informação é transportada por essas mesmas mensagens. Este tipo de sistemas é adequado para a circulação de documentos e dossiers. Possui boas características de escalabilidade e permite a sua utilização em redes de área alargada. O principal problema é a dificuldade em controlar o estado actual do processo, pois as mensagens circulam por espaços de armazenamento privados;
- baseados em bases de dados partilhadas: neste caso, a informação (documental e relativa às tarefas) encontra-se armazenada em bases de dados acessíveis a diversos utilizadores, permitindo, em cada instante, conhecer o estado actual do processo. A sua replicação é aconselhada, tendo em consideração os aspectos relacionados com o controlo de concorrência;
- sistemas cliente-servidor baseados em bases de dados: neste caso, existem servidores que controlam todo o processo, notificando os clientes (elementos do grupo) acerca das tarefas a cumprir, estando a informação armazenada em bases de dados.

Podemos classificar ainda os sistemas *Workflow* de acordo com o grau de estruturação dos seus processos:

- bem estruturado: cada actividade está bem descrita *a priori*, as relações entre as actividades são conhecidas, podem-se definir dependências de informação entre actividades e as responsabilidades são predefinidas. Exemplos típicos deste tipo de sistemas são os sistemas de processamento de vendas;
- semi-estruturado: as actividades estão parcialmente descritas *a priori*, as relações entre as actividades são parcialmente conhecidas, podem-se definir algumas dependências de informação entre actividades e as responsabilidades são parcialmente predefinidas;

- *ad-hoc*: há fracas possibilidades, se não mesmo nenhumas, de descrever *a priori* as actividades, as relações entre elas, as respectivas dependências de informação e a atribuição de responsabilidades. Exemplos típicos deste tipo de sistemas são os sistemas de desenvolvimento de estratégias de Marketing, actividade que depende fortemente do produto e do respectivo público-alvo.

A modelação de sistemas *Workflow* pode ser feita em três vertentes: procedimentos, informação e organização. No primeiro caso, definem-se as actividades e o seu ordenamento; no que diz respeito à informação, é necessário modelar os documentos e objectos manipulados no processo; no último caso, definem-se os intervenientes no processo e as respectivas responsabilidades e permissões dentro do processo. A definição de uma actividade engloba a definição da operação que pretende implementar, da sua estrutura e do contexto de execução (ferramentas necessárias e informação associada ao processo). Uma operação é descrita pela pré-condição (que a desencadeia), pelos dados de entrada e de saída, pela acção a executar e pela pós-condição (despoleta a execução de outra operação).

Existe uma organização normalizadora no âmbito dos sistemas Workflow, designada ***W**ork**f**low **M**anagement **C**oalition* (**WfMC)** [72], fundada em 1993, que tem como objectivo promover o uso deste tipo de sistemas, através do estabelecimento de uma terminologia genérica e de mecanismos de interoperabilidade entre sistemas. O WfMC definiu um **modelo de referência** que engloba diversas ferramentas e serviços e as respectivas interfaces, como pode ser visto na Figura 3-12, bem como uma linguagem de definição de processos – *Workflow Process Definition Language* (**WPDL)** – na qual são descritas as actividades, as suas pré-condições, tipos de dados, recursos utilizados, transições de estado e regras procedimentais. A componente que se encarrega da interoperabilidade entre sistemas *Workflow* é a Interface 4.

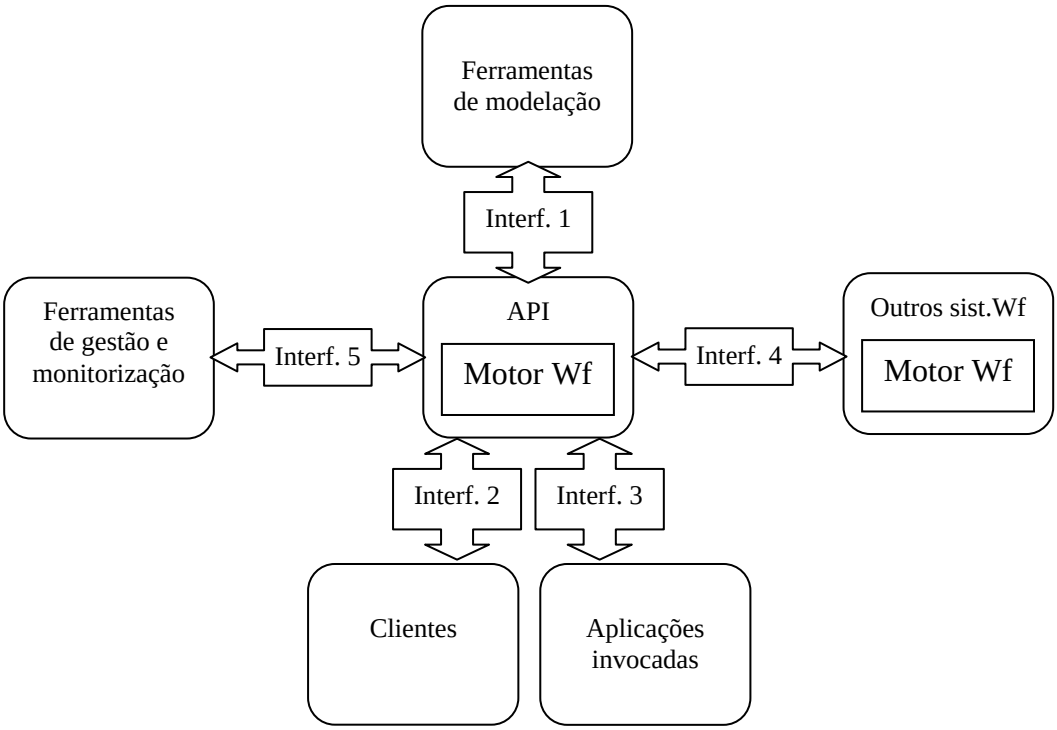

Figura 3-12 – Modelo de referência do WfMC

3.9.4. Suporte a reuniões

Um caso em que se pode beneficiar bastante com a utilização dos computadores como suporte a trabalho de grupo em organizações é o caso das reuniões electrónicas [85], existindo três cenários distintos:

- **reuniões presenciais** ou reuniões face-a-face: neste caso, o computador é envolvido na produção de documentos, em apresentações e pode ainda ser utilizado para a disponibilização de informação a cada um dos participantes. Todos os intervenientes têm um computador à sua disposição, neste último caso (modelo suportado por computador – estrutura da Figura 3-14, com utilizadores e computadores localizados na mesma sala), enquanto que nos dois primeiros casos apenas é disponibilizado ao orador ou ao moderador da reunião (modelo de moderador - Figura 3-13);
- **reuniões distribuídas**: os participantes na reunião encontram-se dispersos geograficamente, exigindo a disponibilidade de meios de conferência e de

áreas de informação partilhada (estrutura da Figura 3-14, com utilizadores e computadores localizados em sítios diferentes);

- **suporte entre reuniões**: utilização do computador para a preparação conjunta de informação a ser usada nas reuniões. São exemplos típicos deste cenário a gestão de agendas de grupo, a gestão de projectos, a edição conjunta de documentos ou mesmo a interacção espontânea usando capacidade multimédia (conceito de **corredores virtuais**, por exemplo).

Note-se que no modelo suportado por computador, quer no caso das reuniões presenciais, quer nas reuniões distribuídas, a componente de moderador pode não existir, possuindo todos os intervenientes os mesmos programas informáticos e telemáticos (existindo, eventualmente, alguém com alguma responsabilidade na condução da reunião).

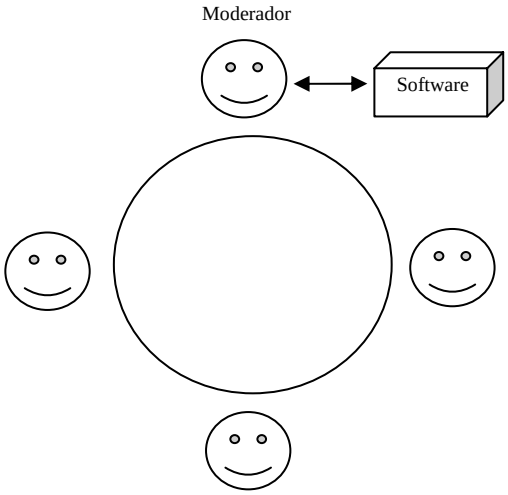

Figura 3-13 – Reuniões electrónicas: modelo de moderador

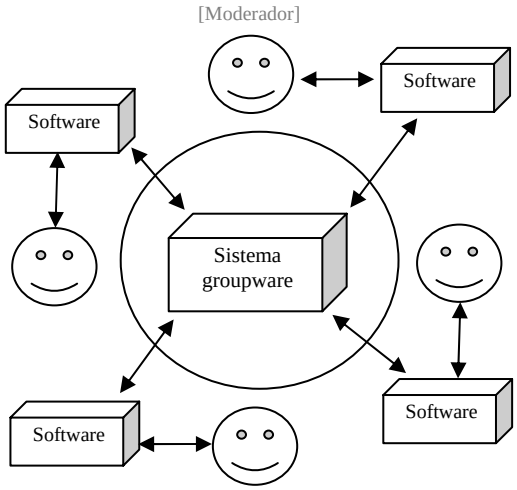

Figura 3-14 – Reuniões electrónicas: modelo suportado por computador

A utilização de meios informáticos e telemáticos como suporte a reuniões permite melhorar a disponibilização de informação aos participantes e minimizar eventuais dificuldades de deslocação dos intervenientes. Para além disso, estes meios podem potenciar a criação ou alteração de esquemas, desenhos e documentos no decorrer da reunião, estimulando a introdução de novas ideias e de diferentes visões acerca das ideias em debate, bem como a detecção de eventuais problemas. O armazenamento em bases de dados de toda a informação produzida no âmbito de uma reunião permite constituir um histórico da reunião em curso e de todas as reuniões efectuadas pelo grupo de trabalho. Este tipo de sistemas permite ainda programar detalhadamente a evolução da reunião, facilitando a sua gestão.

Como em qualquer sistema, também existem inconvenientes na utilização de sistemas de suporte a reuniões electrónicas, como o desvio de atenção relativamente aos assuntos que estão a ser debatidos no momento, a inibição ou frieza provocada pela distância ou a tomada de decisões baseada em informação incompleta (devido ao distanciamento, mais uma vez). Outro problema que pode surgir está relacionado com o relativo anonimato sentido pelos participantes à distância, podendo conduzir a situações demasiado conflituosas, resultantes da acesa discussão de ideias.

A concepção de salas para reuniões presenciais deve levar em consideração a disposição do mobiliário e dos computadores, contemplando aspectos como a visibilidade para um quadro ou uma tela de projecção, ou mesmo o contacto visual entre

participantes. Uma solução típica para o problema da visibilidade e do contacto visual é a integração dos monitores dos participantes na própria mesa de reunião.

Diversos sistemas de suporte a reuniões têm sido desenvolvidos nos últimos anos, como os sistemas "SISCO" [86-88], "NG-TOOL" [89], "LeanMedia" [63] e "RoamWare" [90].

3.9.5. Criação cooperativa de documentos

A criação de documentos envolve tipicamente 3 fases distintas, o planeamento, a edição e a revisão, segundo um processo iterativo. Esta actividade necessita frequentemente da intervenção de diversas pessoas, podendo essa intervenção ocorrer em diferentes zonas do documento, haver necessidade de revisão por parte de editores ou supervisores, ou haver mesmo intervenção simultânea de várias pessoas na mesma zona do documento.

A edição de um documento por diversos intervenientes pode ocorrer em momentos distintos, falando-se, neste caso, de **edição assíncrona**. Esta é a situação mais comum, utilizando-se ferramentas de partilha básicas, como o correio electrónico ou a simples partilha de espaço de armazenamento. À edição simultânea de documentos por diversos utilizadores dá-se o nome de **edição síncrona** e é suportada pelos chamados **editores de grupo** [91-96].

Quando a necessidade de edição cooperativa envolve apenas a supervisão, é suficiente usar ferramentas de edição assíncrona. A utilização de editores de grupo justifica-se nos casos em que há necessidade de intervenção de várias pessoas em várias fases do processo, quer seja em zonas diferentes do documento, quer seja na mesma zona.

A Figura 3-15 mostra a arquitectura genérica de um editor de grupo. Nela se podem ver os módulos presentes nas máquinas dos autores e os módulos respeitantes à unidade de gestão do sistema. O módulo de comunicação permite a comunicação directa entre utilizadores e a comunicação indirecta via unidade de gestão, para propagação de notificações e recolha de dados históricos. O módulo de coordenação global gere dados relativos à actividade dos vários intervenientes, permitindo saber, por exemplo, quem

está a trabalhar e o que está a fazer no momento. A unidade de armazenamento de grupo guarda os documentos a editar, o histórico das actividades do grupo e informação do grupo, como a identificação dos autores ou as tarefas a cumprir por cada um deles. O módulo de controlo de concorrência lida com os aspectos relativos ao acesso simultâneo à informação, fazendo a propagação de notificações de mudanças nos documentos originais e também reflecte as alterações da informação visualizada, de acordo com o paradigma WYSIWIS.

Um dos aspectos importantes em sistemas *groupware* é o da granularidade no acesso à informação partilhada, conforme já foi visto na secção 3.8. Nos editores de grupo usa-se geralmente o parágrafo como elemento de informação partilhada, apresentando uma dimensão adequada à maioria das situações. Outra abordagem também utilizada é a de tornar a granularidade flexível, variando-a com o gosto do utilizador, segundo a estrutura do documento. Isto obriga a que o documento esteja formalmente estruturado, estando referenciados todos os elementos de organização, como as secções, subsecções, tabelas, figuras, etc. Nesta situação, a utilização de *locks* para implementar o controlo de concorrência não se revela adequada, pois um utilizador pode estar a bloquear uma unidade de informação que contém uma outra unidade que já está bloqueada por outro utilizador (secção e subsecção, respectivamente, por exemplo). Neste caso torna-se mais adequado utilizar mecanismos baseados em versões e na notificação de eventos. Outro aspecto muito importante e que também está relacionado com a concorrência é o acto de desfazer uma acção de edição (operação *Undo*), exigindo cuidados especiais para lidar com o problema [97].

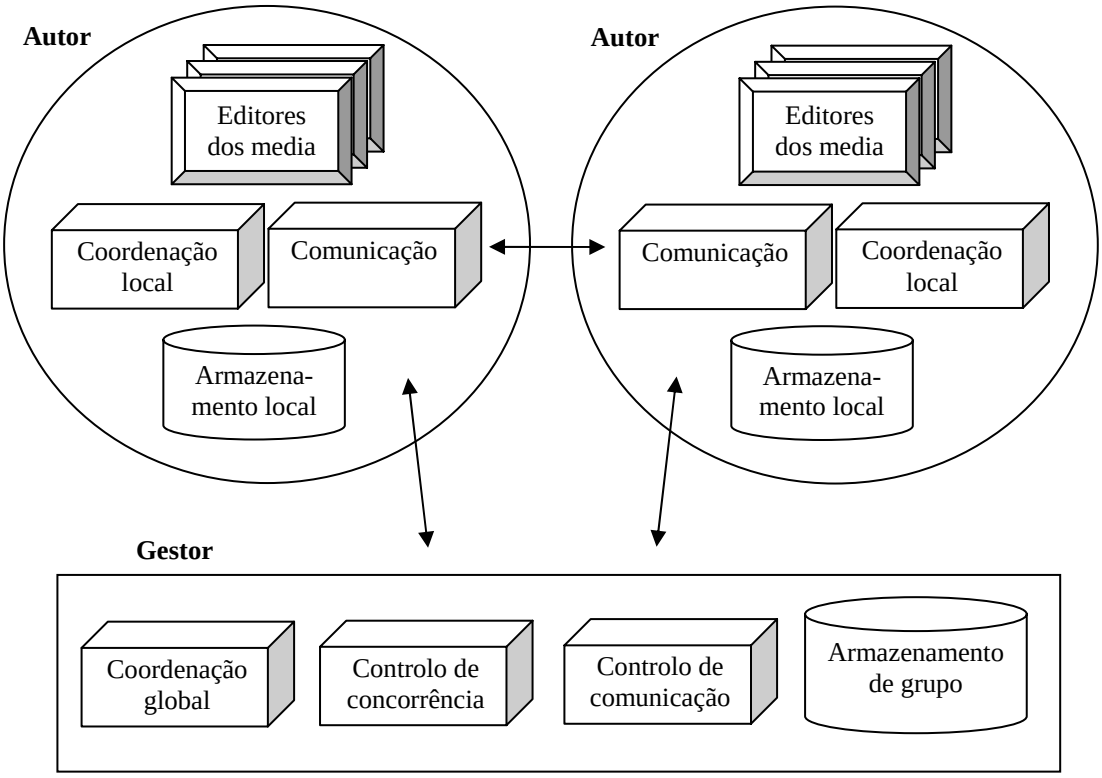

Figura 3-15 – Arquitectura de um editor de grupo

O armazenamento dos documentos a editar pode ser feito de uma forma centralizada (armazenamento num servidor ou na unidade de gestão) ou de uma forma replicada (armazenamento local em cada uma das máquinas dos autores). Uma arquitectura centralizada é adequada em situações de edição onde predomina a escrita sobre a leitura, de forma a garantir a consistência dos dados. Se a leitura prevalecer sobre a escrita, então é preferível uma arquitectura replicada, tornando o acesso à informação mais rápido. Nos editores de grupo existentes costuma predominar a arquitectura replicada, visto o número de acessos para leitura ser previsivelmente muito superior ao de acessos para escrita.

Na última década foram desenvolvidos vários editores de grupo, como o PENCACOLAS [98] e o DistEdit [95] e foram conduzidos diversos estudos acerca da utilização deste tipo de sistemas.

3.9.6. Agentes cooperantes

Genericamente, chama-se agente [99, 100] a um programa que, de alguma forma, ajuda o utilizador nas suas tarefas ou até o substitui em algumas. Os agentes encontram aplicação nas mais variadas actividades, sendo alguns exemplos bem conhecidos, como os presentes nos modernos processadores de texto, que ajudam o utilizador em tarefas como a correcção ortográfica e na prestação de ajuda à utilização do próprio editor. Outra forma bastante comum de utilização de agentes está relacionada com a presença, muitas vezes indesejada e involuntária, de **agentes de monitorização** no computador de um utilizador da Internet, resultantes da instalação de programas gratuitos, e cujo intuito é a recolha de informações pessoais ou de hábitos de trabalho para os mais variados fins (por vezes até ilícitos!). Um tipo de agentes que tem ganho importância nos últimos anos é o dos **agentes móveis**, os quais têm a capacidade de se deslocar de computador para computador autonomamente, segundo um percurso predefinido. A aplicação mais comum deste tipo de agentes é na **procura de informação**, como acontece em alguns motores de busca de informação na Internet.

De uma forma genérica, podem-se classificar os agentes em **activos** ou **passivos**. Estes últimos actuam apenas sob ordens do utilizador, enquanto os primeiros conseguem reagir a estímulos externos e tomar decisões autonomamente. Apesar de os agentes passivos actuarem apenas sob ordens do utilizador, eles continuam a ser autónomos durante a execução.

O facto dos agentes poderem funcionar autonomamente, em maior ou menor grau, traduz-se na possibilidade de possuírem capacidade para dialogar, cooperando na prossecução dos seus objectivos. Os agentes que possuem esta capacidade designam-se **agentes cooperantes** [101], aparecendo muitas vezes referenciados apenas como sistemas multi-agente.

Para além da **autonomia**, um agente deve possuir características de veracidade, benevolência (ausência de intenções conflituosas), racionalidade (o trabalho desenvolvido destina-se à execução de uma tarefa) e reactividade (capacidade de reagir a estímulos externos ou a modificações contextuais). No caso dos agentes activos, eles podem ainda ter as seguintes características:

- proactividade: capacidade de iniciativa – efectuar uma pesquisa de informação ou enviar uma mensagem por iniciativa própria;
- sociabilidade: possibilidade de interagir com outros agentes;
- mobilidade: propagação entre computadores, segundo um percurso predefinido pelo utilizador;
- aprendizagem: capacidade de aprender com o utilizador ou com outros agentes. Por exemplo, um agente pode ir aprendendo os sítios onde pode procurar determinado tipo de informação ou pode aprender os tipos de informação que o seu utilizador procura mais frequentemente;
- capacidade de possuir crenças, desejos e intenções – estas características já se enquadram no âmbito da Inteligência Artificial [102, 103].

Um aspecto importante relativamente aos agentes cooperantes é a linguagem utilizada para comunicarem entre si. Existe uma linguagem, desenvolvida pela *Advanced Research Projects Agency* (ARPA) [104], que define um conjunto de termos e operações que todos os agentes entendem e para as quais podem dar alguma resposta, permitindo ainda a publicitação dos seus serviços. Esta linguagem, designada *Agent Communication Language* (**ACL)**, é composta por três componentes: *Ontolingua*, *Knowledge Interchange Format* (**KIF)** e *Knowlege Query and Management Language* (**KQML)**. A componente *Ontolingua* diz respeito ao vocabulário, o KIF tem a ver com o formato de troca de conhecimentos e o KQML é uma linguagem de pesquisa e manipulação de conhecimentos.

A cooperação entre agentes pode assumir diversas formas:

- acidental: os agentes actuam isoladamente, sem terem a noção de que estão a contribuir para os objectivos de outros agentes;
- *Master-Slave*: um agente delega uma tarefa noutro, que não a pode recusar;
- *one-way*: um agente delega uma tarefa noutro, que tem a opção de a recusar;

- recíproca: ambos os agentes se ajudam mutuamente e ambos podem decidir se querem ou não continuar a cooperar;
- partilha de objectivos: os agentes envolvidos cooperam na execução de tarefas específicas e ambos conhecem os objectivos a atingir.

Aplicações típicas de agentes cooperantes são a procura de informação e a resolução de problemas. No caso dos sistemas de procura de informação é necessário localizar as fontes, pesquisá-las, recolher os resultados da procura e extrair informação útil a partir dos dados recolhidos. Todas estas actividades podem ser efectuadas por diversos agentes que actuam em diferentes fontes ou cada agente pode ter responsabilidades sobre apenas uma tarefa, transmitindo os seus resultados ao agente responsável pela tarefa seguinte.

Nos sistemas de **resolução distribuída de problemas** os agentes cooperam na resolução de um problema, sendo necessário dividir o problema em sub-problemas, os quais serão atribuídos a diferentes agentes. Um exemplo de resolução distribuída de problemas é a **marcação distribuída de reuniões**. Neste tipo de sistemas, o agente do utilizador que pretende marcar uma reunião negoceia com os agentes de outros utilizadores uma data que seja conveniente para todos. A utilização destes sistemas só é viável se todos os utilizadores possuírem e usarem eficazmente um sistema de agendamento electrónico. Na realidade, muitas vezes o processo acaba por se tornar ainda mais trabalhoso, devido à inadaptação de alguns utilizadores.

Um agente cooperante pode ser descrito de acordo com o **Modelo de Insecto**, o qual se encontra esquematizado na Figura 3-16.

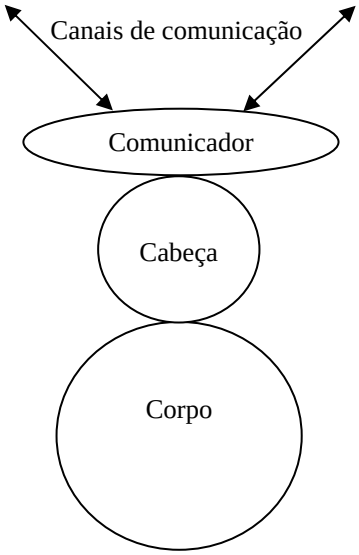

Figura 3-16 – Modelo de Insecto

O Modelo de Insecto divide o agente em três módulos, de acordo com as suas funcionalidades:

- corpo: contém a funcionalidade do agente propriamente dito;
- cabeça: mediador entre a funcionalidade do agente e o contexto de resolução do problema. Este módulo contém informação acerca das capacidades do agente do qual faz parte e também das capacidades dos outros agentes com os quais interage. Para além disso, conhece o problema a resolver, a hierarquia relativa a esse problema e os protocolos de comunicação que deve usar para comunicar com os outros agentes e com o utilizador;
- comunicador: implementa a comunicação com os outros agentes e com o utilizador, através dos canais de comunicação.

Alguns exemplos de implementação de agentes cooperantes são os sistemas "LawBot" [105], "Symgroup" [106] e "GroupLens" [107].

3.9.7. Ensino assistido por computador

Os métodos tradicionais de ensino e formação, de cariz essencialmente presencial, não satisfazem as necessidades de todos os potenciais destinatários, nomeadamente nos casos em que existem problemas de mobilidade, resultantes de deficiências psico-motoras ou da idade avançada (maiores necessidades de conforto, associadas a problemas de saúde, por exemplo). Nestes casos, a disponibilização de meios de ensino e formação à distância revela-se muito importante, permitindo abranger populações tradicionalmente desfavorecidas.

O ensino assistido por computador [108-118] pode ser síncrono ou assíncrono. No ensino assistido por computador assíncrono o educador/formador e os alunos não se encontram presentes simultaneamente no **espaço virtual de ensino**. Enquadram-se nesta categoria as ferramentas de **ensino à distância** baseadas em páginas *web* ou em CD-ROM.

O ensino assistido por computador síncrono pressupõe a presença simultânea do educador e dos alunos num mesmo espaço, real ou virtual. Dentro da categoria do ensino assistido por computador síncrono existem dois tipos: a **aula electrónica** e a **aula virtual**. Na aula electrónica, quer o educador, quer os alunos, encontram-se presentes na mesma sala ao mesmo tempo, sendo usado o computador como uma ferramenta de apoio didáctico. Na aula virtual, o educador e os alunos encontram-se dispersos por diversos sítios, servindo o computador para efectuar a comunicação entre os intervenientes e também como ferramenta de apoio didáctico.

3.9.8. Ambiente Cooperativo Virtual

Existem sistemas *groupware* que utilizam técnicas de realidade virtual e modelação 3D para representarem elementos activos ou passivos do processo de trabalho ou para ajudarem na execução das tarefas. Alguns sistemas de telepresença usam **representações poligonais 3D** de objectos ou mesmo para representarem as pessoas envolvidas, como forma de poupar recursos de transmissão ou para adicionar um cenário virtual, com elementos decorativos, a uma imagem limitada à face do

interlocutor. Este tipo de sistemas é habitualmente referenciado como *Collaborative Virtual Environment* (CVE). Um exemplo de aplicação CVE é o sistema "JCAD-VR" [119], que permite a colaboração entre Arquitectos usando representações tridimensionais.

Um tipo importante de sistemas *groupware* que utilizam realidade virtual é o que habitualmente se designa por **sistemas de imersão**. Nestes sistemas tenta-se criar um ambiente à volta do utilizador que lhe crie a ilusão de estar localizado num sítio remoto ou junto dos seus interlocutores [120]. Estes sistemas utilizam esquemas complexos de visualização, recorrendo a múltiplos monitores dispostos de uma forma envolvente (Figura 3-17). Por exemplo, no caso de uma aplicação de suporte a reuniões, pode-se criar a ilusão de continuidade da mesa do utilizador com as mesas dos interlocutores.

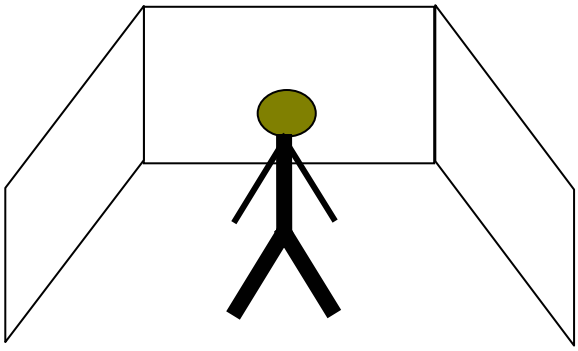

Figura 3-17 – Sistema de Imersão

Um exemplo de aplicação de imersão é o sistema de visitas partilhadas "Shared visiting in EQUATOR city" [121], que permite a visita conjunta de visitantes reais e virtuais.

Outro caso interessante de utilização de realidade virtual é o dos sistemas de **telemanipulação**. Neste tipo de sistemas, existem objectos que podem ser manipulados sincronizadamente por vários participantes remotos, criando a ilusão de interacção. Esses objectos são geralmente interfaces palpáveis [122], como luvas de realidade virtual. Num exemplo deste tipo de sistemas [62], faz-se a manipulação remota de amostras biológicas, usando um dispositivo chamado nanomanipulador, para serem estudadas num microscópio de força atómica.

3.10. A *web* como suporte à cooperação

A Internet faz cada vez mais parte do quotidiano de muitas pessoas, principalmente dos profissionais de investigação e de meios empresariais, os quais utilizam frequentemente o correio electrónico e a *web* como ferramentas de trabalho.

As aplicações *groupware* dependem fortemente, na sua maioria, de meios gráficos de visualização e da correcta adequação dos seus utilizadores à aplicação. A utilização de interfaces *web* em aplicações cooperativas (recorrendo a *applets*), pode ser particularmente atractiva, quer do ponto de vista visual, quer do ponto de vista de integração de diversos media, quer ainda pela familiarização dos utilizadores com este tipo de interfaces gráficas. Existem, contudo, dificuldades relacionadas com a eficaz comunicação entre os intervenientes (velocidade, fiabilidade, comunicação entre *applets*) e com os aspectos de segurança e privacidade, as quais necessitam de um tratamento adequado, recorrendo a tecnologias adicionais que permitam solucionar esses problemas. Os sistemas *groupware* "QUORUM-W" (suporte à decisão) [123], "Social Web Cockpit" (recomendações) [77], "WW-FLOW" (*Workflow*) [124] e "WWG" (ensino) [115] usam a *web* como meio de suporte à cooperação.

Capítulo 4

Um Modelo para a Criação de Serviços Cooperativos

Este capítulo apresenta um modelo para a criação de aplicações de trabalho cooperativo suportado por computador, utilizando *web services* como plataforma de comunicação distribuída em ambientes potencialmente heterogéneos. Apresentam-se as motivações para a adopção deste modelo e a sua arquitectura genérica e, de seguida, faz-se uma apresentação mais detalhada das diversas componentes da arquitectura. Finalmente, discutem-se alguns cenários de utilização do modelo proposto.

4.1. Introdução

As metodologias de trabalho nas organizações são cada vez mais diversas e complexas e as responsabilidades atribuídas a cada interveniente no processo nem sempre estão definidas de uma forma estática. Assim, é frequente que uma dada tarefa seja executada recorrendo a diversas pessoas, as quais podem, inclusivamente, não ter sempre o mesmo papel na execução de tarefas do mesmo tipo em alturas distintas. Para além disso, durante a execução de uma dada tarefa, uma pessoa pode necessitar de consultar outras, para se aconselhar sobre alguns aspectos específicos ou para obter aprovação por parte de instâncias superiores da organização.

As aplicações de ***groupware*** tentam suportar da melhor maneira possível os aspectos relacionados com o **trabalho de grupo**, fornecendo meios para a partilha de informação, para a comunicação entre os intervenientes numa determinada tarefa e para a manipulação conjunta da informação e/ou de meios de tratamento dessa informação. Contudo, o modelo habitual de desenvolvimento de aplicações cooperativas assenta fundamentalmente em soluções proprietárias, pouco flexíveis e sem preocupações de interoperabilidade e/ou de aproveitamento de sistemas já existentes.

A utilização das aplicações *groupware* nem sempre é intuitiva ou se encontra dentro dos padrões de utilização de conhecimento generalizado, implicando, frequentemente, que o utilizador se adapte a ferramentas radicalmente diferentes daquelas que está habituado a usar. Este factor contribui, com alguma frequência, para uma fraca ou lenta aceitação deste tipo de aplicações, com todos os custos associados a esse processo de rejeição.

A Internet tornou-se praticamente omnipresente nos meios organizacionais e a generalidade dos utilizadores de redes informáticas usa massivamente o correio electrónico e a ***Web***, estando bastante familiarizados com as respectivas ferramentas. Desta forma, o ambiente gráfico e os paradigmas de utilização das ferramentas comuns da Internet adequam-se bem a uma habituação rápida e progressiva a novas aplicações. A tecnologia **Java** veio facilitar a construção de aplicações que tirem partido destas características, mas por si só não consegue assegurar diversos aspectos relacionados

com a segurança e privacidade das comunicações, nem garante a interoperabilidade com outros sistemas desenvolvidos recorrendo a diferentes linguagens de programação.

A comunidade científica internacional, com apoios importantes da indústria e dos meios académicos, fez um esforço considerável de normalização da comunicação em ambientes heterogéneos através da arquitectura **CORBA**, mas esta não se conseguiu afirmar como uma solução amplamente consensual, devido à sua complexidade, que levou a que muitos serviços não fossem implementados, e ao facto de alguns fabricantes adoptarem soluções proprietárias em certos serviços e/ou funcionalidades da infra-estrutura, reduzindo a sua interoperabilidade.

As limitações da arquitectura CORBA e o aparecimento da linguagem XML [38, 125], tornaram viável o desenvolvimento, no seio do W3C (*World Wide Web Consortium*) [34], de uma nova tecnologia de suporte à comunicação distribuída, conhecida por *web services*. Nesta tecnologia, a utilização do XML para a descrição dos serviços e para a troca de mensagens confere-lhe independência relativamente à linguagem de programação, permitindo a tão desejada interoperabilidade entre aplicações escritas em linguagens de programação distintas. Para além disso, a utilização do XML permite a adição de metadados, que podem conferir uma maior riqueza semântica às aplicações. Acresce a tudo isto que as infra-estruturas de *web services* dispõem de diversos serviços de suporte ao desenvolvimento de aplicações e de serviços que lhes oferecem mecanismos de segurança. Os *web services* podem ainda ser usados para adaptar sistemas já existentes, transformando-os em serviços disponíveis *online*, rentabilizando investimentos já feitos e reduzindo o tempo de desenvolvimento ou de disponibilização de novos serviços.

O desenvolvimento de aplicações cooperativas baseadas na *web* pode mostrar-se uma opção bastante atractiva pela familiaridade que o respectivo ambiente oferece à maioria dos utilizadores, mais ou menos frequentes, das redes informáticas organizacionais. A utilização de *web services* no suporte às funcionalidades deste tipo de aplicações permite tirar partido do potencial de distribuição que a tecnologia oferece, nomeadamente no que diz respeito aos aspectos de **interoperabilidade**, de **segurança** e de reaproveitamento de **sistemas legados**. Para além disso, trata-se igualmente de uma tecnologia fortemente suportada pela indústria e pela comunidade académica.

A utilização dos *web services* em aplicações cooperativas tem mais expressão no caso dos sistemas de ***workflow*** [126], existindo mesmo algum trabalho de normalização nesse campo. De facto, a especificação **Wf-XML**, da WfMC (*Workflow Management Coallition*) [72], define um formato de mensagens XML para interoperabilidade entre sistemas de *workflow*. Contudo, esta é a única categoria de aplicações cooperativas em que existe trabalho normativo. De uma forma geral, os *web services* são usados essencialmente em aplicações transaccionais de negócio, não se encontrando tão bem documentada a sua utilização noutros tipos de aplicações cooperativas.

Neste capítulo desta Tese de Doutoramento define-se um modelo de criação de aplicações cooperativas, usando *web services* como suporte à comunicação entre as diversas componentes do sistema, seja para a invocação de serviços de suporte, seja para a notificação dos eventos relacionados com a partilha de informação e/ou de funcionalidades entre as partes cooperantes.

4.2. Modelo proposto

As actividades cooperativas incorporam, geralmente, pelo menos uma das três funcionalidades seguintes:

- comunicação entre elementos do grupo de trabalho;
- partilha de informação;
- visualização conjunta de actividades ou ambientes de trabalho.

Para as duas últimas funcionalidades pode ser importante garantir o acesso exclusivo aos recursos envolvidos, pelo que é desejável que exista um mecanismo que permita efectuar um eficaz **controlo de concorrência**, ora permitindo que vários processos manipulem um determinado recurso livremente, ora impondo limitações à sua manipulação, ou até mesmo restringindo o seu controlo a um único processo (através de ***locks***, por exemplo). Para além disso, para que um utilizador possa visualizar acontecimentos provocados pelas actividades dos restantes, deverá haver um mecanismo de **notificação de eventos** que se encarregue da distribuição dessas acções.

Outro aspecto importante das aplicações cooperativas é o facto de se dirigirem a grupos limitados de utilizadores. De facto, uma dada aplicação cooperativa ou conjunto de aplicações cooperativas destina-se a ser usada por elementos de uma equipa de trabalho ou de uma organização, os quais se encontram perfeitamente identificados. Mesmo em casos de utilização inter-organizacional, essa utilização encontra-se igualmente bem identificada e restringida a determinado grupo de utilizadores. Desta forma, o sistema cooperativo deve possuir um mecanismo de **autenticação** dos utilizadores.

A cooperação pode assumir um carácter esporádico ou podem mudar com alguma frequência as tarefas atribuídas a cada utilizador envolvido nas actividades cooperativas. Este aspecto torna atractiva a possibilidade de dispor de um **repositório de aplicações cooperativas**, ao qual os utilizadores acedem para escolher as aplicações de que necessitam num determinado momento para a prossecução das tarefas que lhe estão atribuídas. Esta abordagem tem a vantagem de possibilitar que cada utilizador disponha sempre da versão mais actualizada de cada aplicação, criando um ambiente cooperativo homogéneo.

Por fim, cada utilizador deve poder conhecer, em cada momento, a lista de utilizadores que estão registados, de forma a poder iniciar uma sessão cooperativa com um dado utilizador ou grupo de utilizadores. Assim, deverá existir um **directório de utilizadores** registados, que mostra a lista de utilizadores do sistema e as actividades em que se encontram envolvidos.

Pelos motivos apontados nos últimos parágrafos, um sistema genérico de aplicações cooperativas deve possuir o seguinte conjunto de funcionalidades:

- Serviço de Autenticação – para controlar o acesso ao sistema;
- Serviço de Directório de Utilizadores – para que os utilizadores saibam com quem podem cooperar;
- Serviço de Repositório de Aplicações – que fornece uma lista de aplicações disponíveis para carregamento a pedido para o computador do utilizador;
- Serviço de Repositório de Informação – um sistema de ficheiros e/ou base de dados que guarde os diversos recursos informativos, sejam eles ficheiros de texto, informação multimédia ou metadados;

- Serviço de Controlo de Concorrência – para implementar mecanismos que permitam controlar o acesso aos recursos informativos, podendo até garantir o acesso exclusivo a um dado componente do sistema;
- Ferramentas de comunicação de grupo – para implementar a comunicação entre elementos do grupo, de uma forma textual e/ou audiovisual;
- Serviço de Notificação de Eventos – que permite a distribuição de eventos relativos a modificações de contexto em ambientes partilhados.

As funcionalidades referidas como serviços podem ser implementadas recorrendo a *web services*, conferindo ao sistema a desejável interoperabilidade entre aplicações que possam estar implementadas em linguagens de programação distintas, bem como a possibilidade de aproveitar sistemas legados. No restante desta Tese passará a ser usado o termo serviços para identificar os *web services*, por uma questão de simplificação de linguagem.

A Figura 4-1 mostra as diversas entidades presentes num **sistema genérico de aplicações cooperativas** e as actividades em que essas entidades se envolvem. As entidades que correspondem a serviços estão devidamente identificadas.

O Serviço de Repositório de Aplicações permite ao utilizador, após ter sido autenticado no sistema pelo serviço de autenticação, visualizar a lista de aplicações que se encontram disponíveis. O utilizador pode então seleccionar múltiplas aplicações e carregá-las, caso não possua as suas últimas versões. Após a recepção das aplicações, estas serão automaticamente activadas e o utilizador pode começar a usá-las de imediato. Estas aplicações, durante a sua execução, poderão recorrer a alguns dos outros serviços disponíveis.

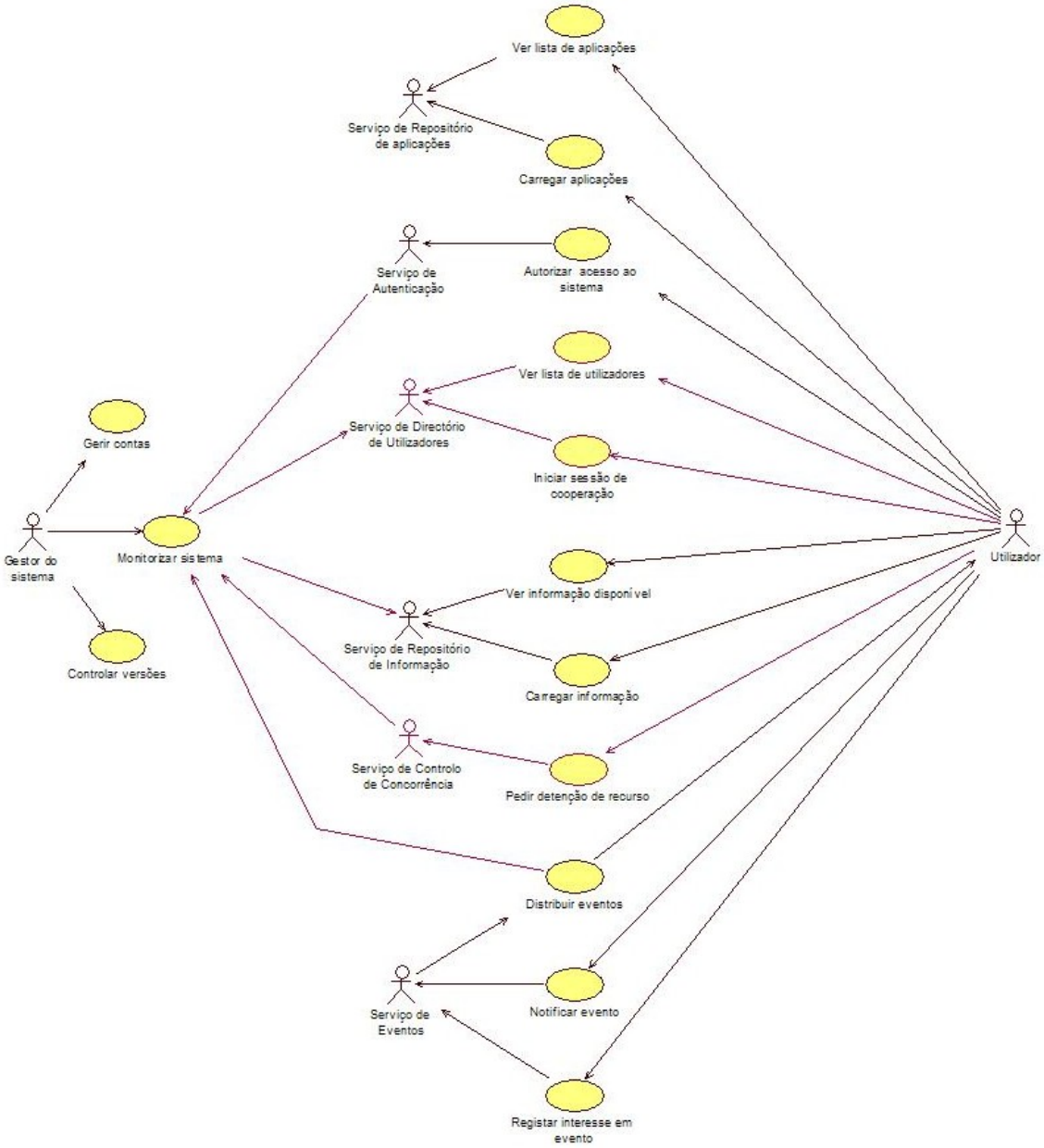

Figura 4-1 – Modelo de utilização de aplicações cooperativas

No Serviço de Repositório de Informação, o utilizador pode visualizar os recursos informativos disponíveis e carregar para o seu computador os que desejar. Os recursos de informação podem ser ficheiros com informação textual ou multimédia ou metadados. Se for necessário garantir a consistência de dados no acesso a um determinado recurso, o cliente pode pedir ao serviço de controlo de concorrência que lhe conceda um mecanismo que lhe garanta acesso exclusivo ao recurso (*lock*, por exemplo), sendo-lhe esse mecanismo concedido caso tal seja possível.

Quando um utilizador deseja cooperar com outro utilizador do sistema, acede ao serviço de directório de utilizadores, para ver com quem poderá estabelecer uma sessão de actividades cooperativas. Esta sessão pode envolver apenas a utilização de recursos de comunicação, ou pode também recorrer à partilha de recursos informativos ou aplicativos. Caso haja partilha de recursos, o cliente regista, no serviço de eventos, o interesse em ser notificado acerca da ocorrência de eventos relativos aos recursos partilhados.

O modelo proposto é um modelo distribuído replicado, em que as diversas actividades de grupo se encontram distribuídas por vários computadores, sendo algumas repetidas em vários computadores.

No modelo proposto, existe uma entidade, o **gestor do sistema**, que controla toda a sua actividade. Das suas atribuições fazem parte a criação e a gestão das **contas** dos utilizadores, o controlo de **versões** dos serviços e das aplicações disponíveis e a **monitorização** de todo o sistema. Da monitorização fazem parte as seguintes tarefas:

- controlo de acesso ao sistema;
- gestão do directório de utilizadores;
- gestão da informação disponível nos repositórios de informação e de aplicações;
- monitorização dos eventos ocorridos no sistema;
- monitorização dos recursos sujeitos a controlo de concorrência.

O sistema de aplicações cooperativas do modelo proposto possui a arquitectura genérica que se encontra ilustrada na Figura 4-2. Nela pode-se ver que a arquitectura segue o modelo cliente-servidor, encontrando-se do lado do servidor os diversos serviços que o sistema usa para a implementação das funcionalidades cooperativas e do lado do cliente os múltiplos clientes que se ligam ao sistema. As aplicações, a informação a disponibilizar aos utilizadores (multimédia, metadados e listas) e a informação de sistema (resultante da monitorização) encontram-se armazenadas no que é designado por armazenamento de grupo. Porém, cada cliente armazena localmente as aplicações que vai buscar ao repositório de aplicações, bem como ficheiros multimédia que carrega a partir do repositório de informação.

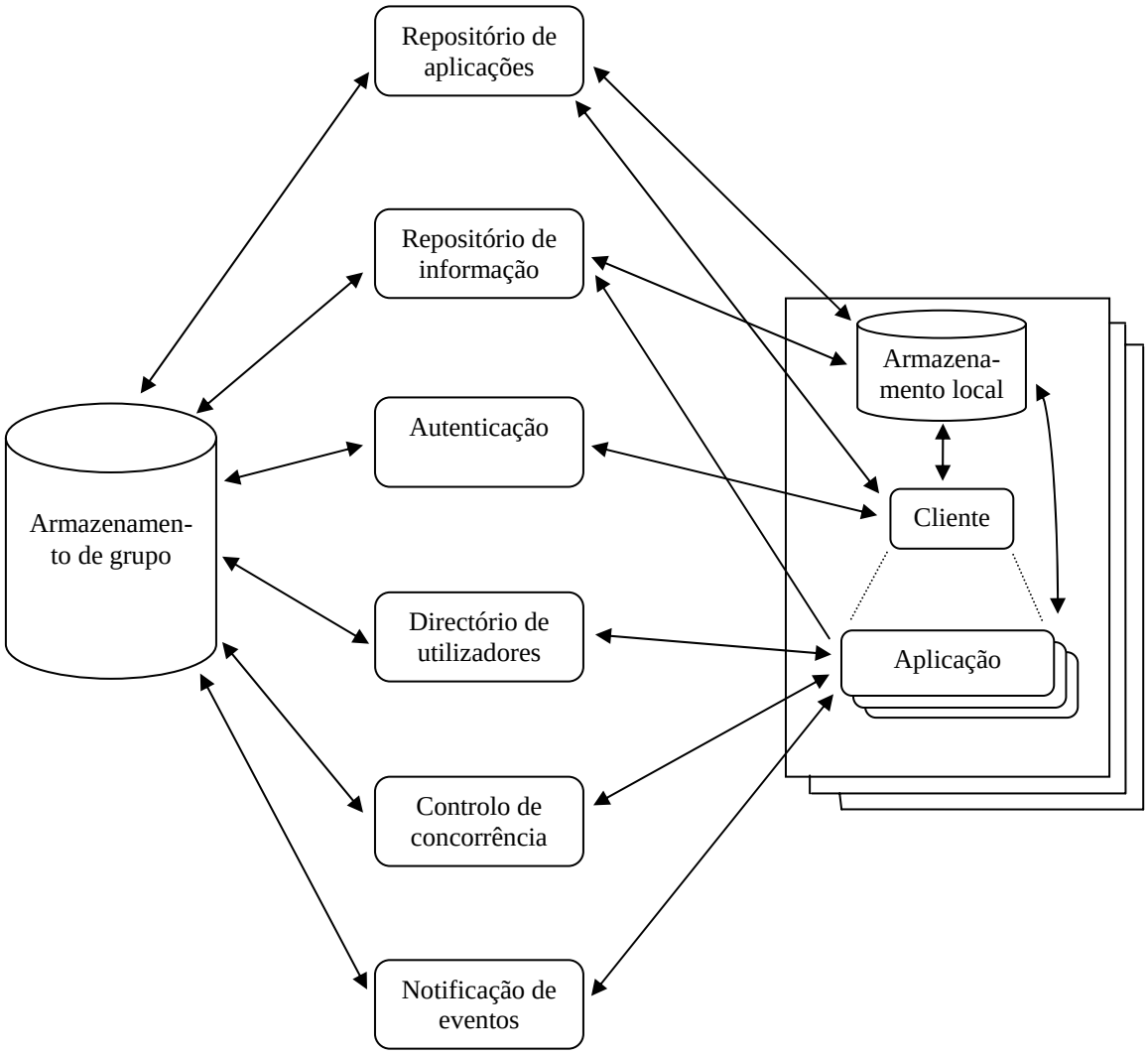

Figura 4-2 – Arquitectura genérica do sistema de aplicações cooperativas

A arquitectura que se apresenta possui as seguintes características:

- aberta – não se baseia em soluções proprietárias, seguindo normas e protocolos amplamente suportados;
- distribuída – os serviços podem estar dispersos por diversos computadores, mantendo agrupado um conjunto nuclear de serviços;
- interoperável – permite a coexistência de aplicações e serviços desenvolvidos e residentes em diversas plataformas computacionais;
- actualizável – o carregamento das aplicações permite dispor sempre da última versão de cada aplicação;

- modular – as diversas funcionalidades encontram-se bem identificadas e dispersas por diferentes aplicações e serviços, apenas se carregando e usando as que são necessárias;
- evolutiva – é fácil adicionar novas funcionalidades ao sistema, quer através da disponibilização de novos serviços, quer pela colocação de novas aplicações no respectivo repositório. Apresenta ainda a possibilidade de aproveitamento de sistemas legados;
- segura – as infra-estruturas de *web services* possuem mecanismos de segurança, nomeadamente de autenticação e de encriptação.

4.3. Aplicações cliente do sistema

As aplicações cliente de *web services* não interagem directamente com os serviços de que necessitam. De facto, a comunicação é intermediada por *proxies*, que se encontram na máquina do cliente e propagam as invocações aos respectivos serviços. Assim, o cliente do sistema de aplicações cooperativas utiliza *proxies* para os serviços de autenticação e de repositório de aplicações, instanciando-os quando necessário.

A Figura 4-3 ilustra o funcionamento de um cliente do sistema de aplicações cooperativas e a forma como ele usa os serviços de autenticação e de repositório de aplicações. Nela é visível que, logo após o início da sua execução, o cliente cria uma *proxy* do Serviço de Autenticação. Posteriormente, utiliza essa *proxy* para pedir autorização de entrada no sistema, após ter introduzido a sua identificação e senha de acesso. Caso falhe a autenticação, o utilizador dispõe de mais um número predeterminado de tentativas e, caso ultrapasse esse número sem conseguir obter a autorização de acesso, é encerrada a aplicação do cliente.

Depois de o utilizador ter sido autenticado, o cliente instancia a *proxy* do Serviço de Repositório de Aplicações e invoca um método que lhe devolve a lista de aplicações disponíveis. Nesta lista, o utilizador escolhe as aplicações que deseja utilizar e inicia o carregamento de cada uma delas, caso não esteja disponível no disco local ou, apesar de já estar disponível uma versão local, caso esta não seja a última versão conhecida. No caso de já estar disponível localmente a última versão de uma aplicação, não se procede ao seu carregamento e, deste modo, não se perde tempo desnecessariamente com o

processo de transferência dos ficheiros contendo as aplicações. Após o carregamento das aplicações, estas são imediatamente executadas.

A possibilidade de carregamento das aplicações em tempo de execução proporciona uma maior flexibilidade no que diz respeito aos programas instalados no computador de cada utilizador, podendo assim haver uma melhor gestão do armazenamento local, principalmente quando as aplicações não são usadas com frequência. A utilização pouco frequente de uma aplicação aumenta a probabilidade de ela não se encontrar actualizada na altura em que é necessária, sendo este mais um motivo para proceder ao seu carregamento, passando-se a dispor da versão mais recente.

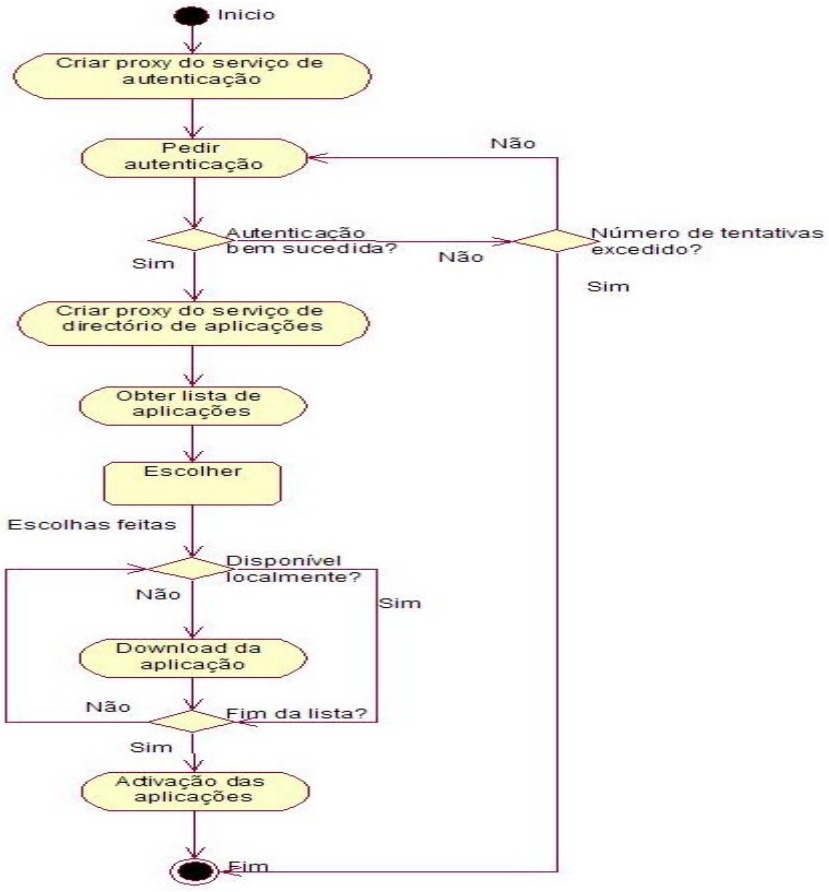

Figura 4-3 – Diagrama de actividade do cliente

A Figura 4-4 mostra a arquitectura interna do cliente do sistema de aplicações cooperativas, bem como as suas ligações externas com os serviços de que necessita. A seta a tracejado indica que a interacção entre o cliente e o servidor para a transferência

de ficheiros só se verifica no caso de ser necessário, o que nem sempre acontece, como já foi explicado anteriormente.

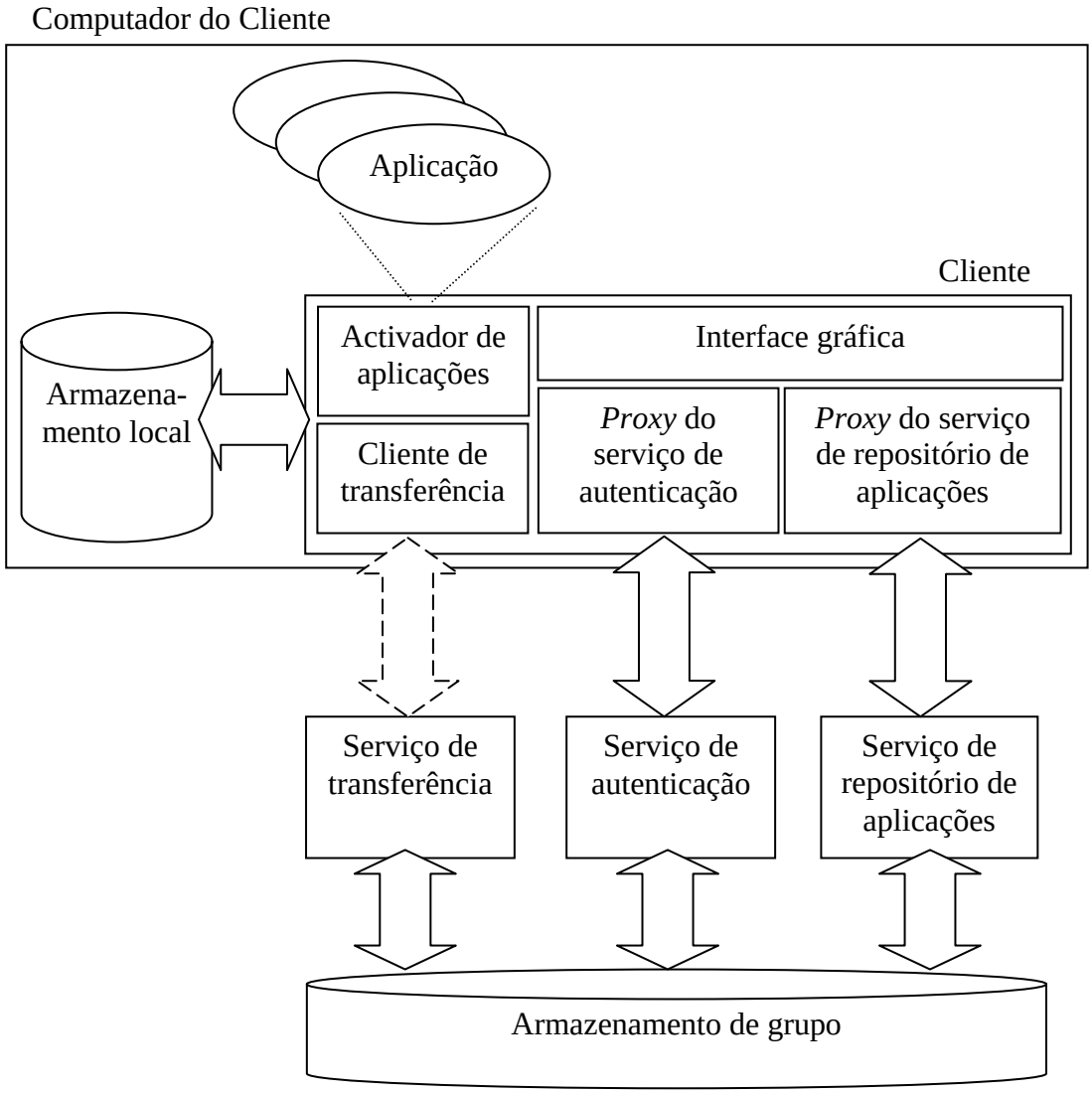

Figura 4-4 – Arquitectura do Cliente

4.4. Serviço de autenticação

A Figura 4-5 mostra a arquitectura do Serviço de Autenticação, bem como as suas ligações a outras componentes do sistema de aplicações cooperativas. A referida arquitectura engloba o *web service* que os clientes utilizam para aceder ao serviço de

autenticação, uma aplicação de gestão das contas dos utilizadores e a base de dados que guarda a informação dessas contas (identificação e senha).

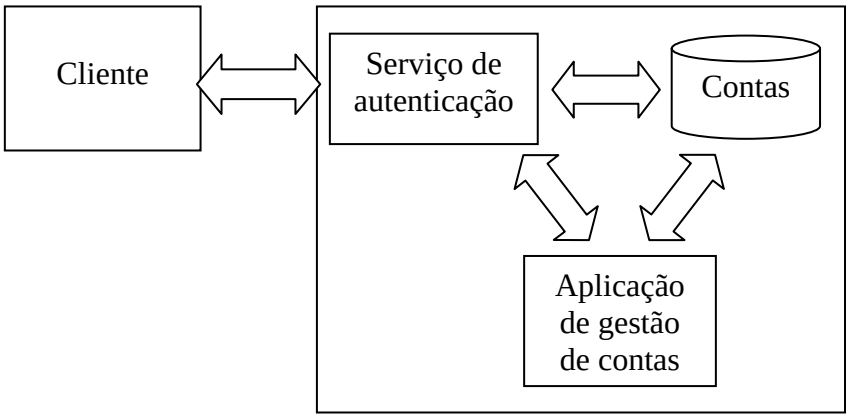

Figura 4-5 – Arquitectura do Serviço de Autenticação

A aplicação de gestão de contas só pode ser utilizada pelo gestor do sistema, razão pela qual também usa o serviço de autenticação. A gestão de contas permite adicionar um novo utilizador ou remover um já existente. A sua actividade está esquematizada na Figura 4-6. A adição de uma nova conta de utilizador implica a inserção de um identificador (nome/*login*) e de uma senha de acesso, a qual necessita de confirmação, como é usual nos vulgares sistemas de gestão de contas de utilizadores. A remoção de uma conta apenas envolve a afixação de uma mensagem de confirmação e a respectiva aceitação.

Quando um utilizador é autenticado, o serviço de autenticação faz o seu registo no directório de utilizadores, de forma que os outros utilizadores possam ter conhecimento da sua actividade e, possivelmente, iniciar com ele uma sessão cooperativa.

A interface gráfica da aplicação de gestão de contas possui uma janela principal onde é possível escolher uma das duas opções já descritas, adicionar ou remover conta, a par da opção de terminar a aplicação.

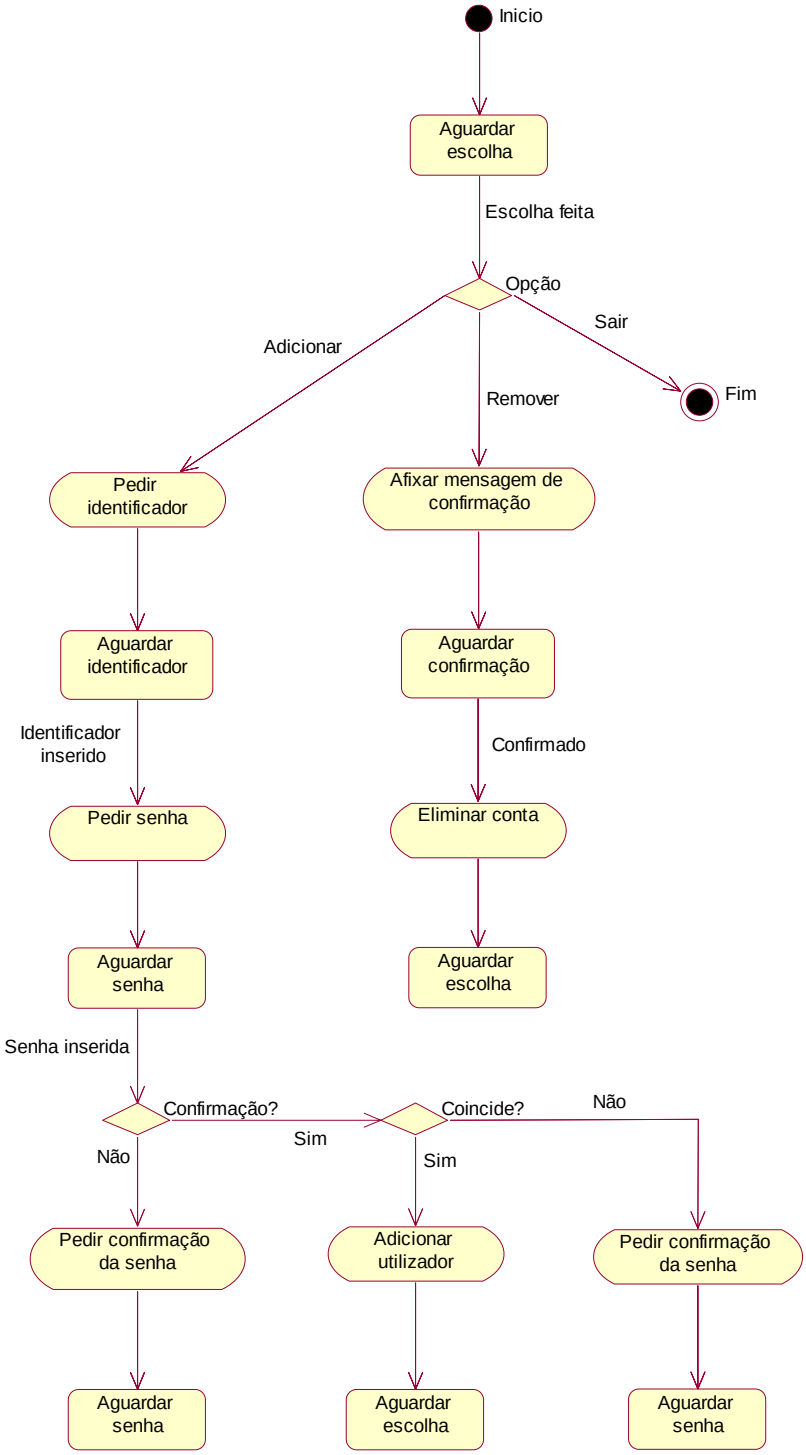

Figura 4-6 – Diagrama de actividade da aplicação de gestão de contas

O *web service* que implementa o Serviço de Autenticação apenas possui uma operação invocável pelos clientes. Essa operação permite validar a identificação e a senha inseridas pelo utilizador na aplicação cliente, as quais vai verificar na base de dados das contas. Se o par identificador/senha for válido, o serviço retorna o valor lógico verdadeiro, caso contrário retorna falso. Trata-se, assim, de um serviço cuja interface é bastante simples, sendo facilmente usada por qualquer tipo de cliente, construído em qualquer linguagem de programação. Toda a complexidade que possa existir relacionada com a gestão e validação das contas está escondida da interface acessível aos clientes. Note-se ainda que a gestão das contas se encontra separada do serviço de autenticação, o que permite reutilizar sistemas de gestão de contas já existentes, apenas sendo necessário construir o serviço.

4.5. Serviço de Directório de Utilizadores

O Serviço de Directório de Utilizadores fornece meios para que estes consigam descobrir quais os outros utilizadores com quem se podem envolver em sessões de trabalho cooperativo. Para que tal seja possível, o directório de utilizadores mantém informação acerca dos utilizadores do sistema de aplicações cooperativas que se encontram activos e quais as aplicações que cada um deles está a utilizar. Desta forma, cada utilizador pode aceder ao serviço de directório, consultar esta informação e notificar um dado utilizador acerca do seu interesse em estabelecer uma sessão de actividade conjunta, partilhando informação e/ou ambientes gráficos.

Cada utilizador é registado no directório de utilizadores aquando da sua autenticação perante o sistema de aplicações cooperativas. De facto, só após a sua autenticação ter sido bem sucedida é que é notificado o directório de utilizadores acerca de um novo utilizador que acabou de entrar no sistema. Após o registo do utilizador, este passa a poder utilizar directamente o serviço de directório para obter a informação necessária ao estabelecimento de sessões cooperativas. O registo das actividades em que cada utilizador se encontra envolvido é feito pelas respectivas aplicações directamente no serviço de directório de utilizadores.

Na Figura 4-7 encontram-se esquematizadas as ligações que o serviço de directório de utilizadores possui com as outras componentes do sistema de aplicações cooperativas. Nela pode-se ver a ligação do cliente com o serviço de autenticação, para fazer a autenticação do utilizador, bem como a ligação com as aplicações, para que estas possam fazer o registo da actividade do utilizador, a pesquisa de potenciais parceiros para o estabelecimento das sessões cooperativas e a remoção do directório, antes de sair do sistema. Toda a informação do directório pode ser armazenada numa base de dados onde é guardada informação acerca dos utilizadores.

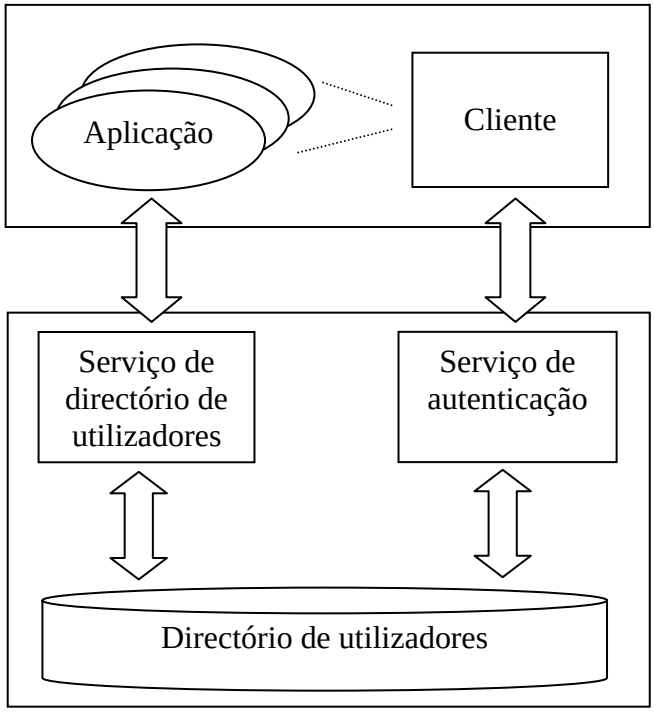

Figura 4-7 – Ligações do Serviço de Directório de Utilizadores

O *web service* que implementa o Serviço de Directório de Utilizadores disponibiliza, na sua interface de acesso, as seguintes operações que os seus clientes podem invocar:

- registo de utilizador – invocada após a autenticação do utilizador, para efectuar o seu registo perante o sistema;
- registo de actividades dos utilizadores – invocada pelas aplicações para registarem o seu início de actividade;

- listagem de utilizadores – invocada pelas aplicações para descobrirem quais os utilizadores que se encontram registados, de forma a poderem encontrar os parceiros cooperativos;
- remoção de uma actividade – invocada pelas aplicações quando pretendem terminar uma actividade cooperativa;
- remoção de utilizador – invocada pelo cliente quando o utilizador abandona o sistema.

4.6. Serviço de Repositório de Aplicações

O Serviço de Repositório de Aplicações permite aos utilizadores do sistema de aplicações cooperativas obter uma lista das aplicações disponíveis e escolherem nessa lista as aplicações que desejam utilizar, fazendo o seu carregamento, caso não disponham localmente da versão mais actualizada, ou não disponham mesmo de nenhuma versão. Na Figura 4-8 podem-se ver as ligações existentes entre o serviço de repositório de aplicações e as componentes do sistema com as quais interage directamente.

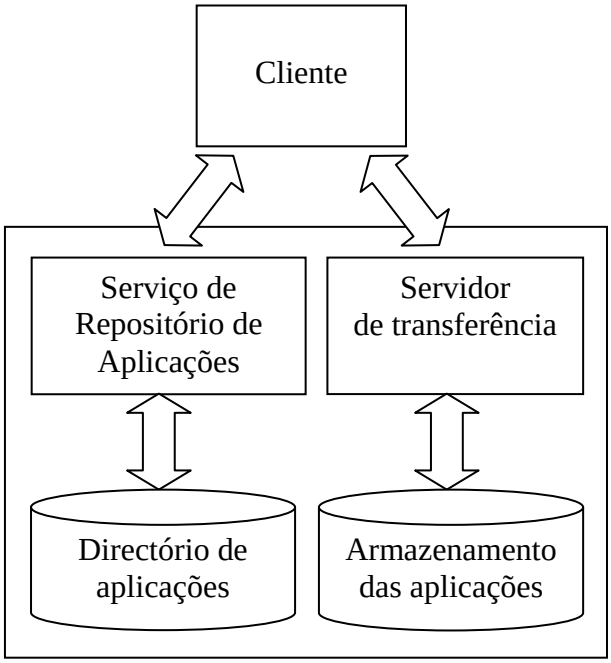

Figura 4-8 – Ligações do Serviço de Repositório de Aplicações

O servidor de transferência é utilizado quando o utilizador pretende proceder ao carregamento das aplicações que escolheu, após ter consultado a lista. O armazenamento de grupo guarda todas as aplicações disponíveis para carregamento por parte dos clientes.

O Serviço de Repositório de Aplicações precisa de construir a lista de aplicações a que cada cliente tem acesso. Basicamente, essa lista pode ser construída segundo dois métodos:

- simples análise da directoria onde se encontram armazenadas as aplicações, devolvendo a lista dos nomes dos ficheiros das aplicações;
- criação de uma base de dados de metadados, onde, para cada aplicação, se guarda o seu nome e informação diversa, como a versão ou uma breve descrição.

A existência dos metadados é particularmente importante no caso do controlo de versões das aplicações. O primeiro método apresentado para a construção da lista é de implementação bastante simples e o seu desempenho deverá ser mais rápido, pois a quantidade de informação a processar é menor do que no caso de se utilizar metadados. Contudo, tem a desvantagem de não permitir ao utilizador obter mais informação acerca das aplicações disponíveis. Neste caso, será necessário recorrer a um qualquer mecanismo que permita aos utilizadores fazer o controlo de versões, como, por exemplo, a manutenção de uma página com essa informação, actualizada pelo gestor do sistema. Outro mecanismo possível é a comparação da data da última alteração dos ficheiros, o que aumenta a complexidade do lado do cliente.

O segundo método de construção da lista de aplicações é de implementação mais complexa, mas é também uma solução mais completa e versátil, pois permite aos utilizadores conhecerem *a priori* muito mais acerca de cada uma das aplicações, antes de proceder ao seu carregamento e subsequente utilização. Cabe, mais uma vez, ao gestor do sistema manter actualizada toda a informação guardada na base de dados, à qual se dá o nome de directório de aplicações. A informação guardada no directório de aplicações pode ser cruzada com a informação guardada no directório de utilizadores,

permitindo aos utilizadores saber quem está a utilizar determinada aplicação, aquando da visualização da lista, o que também pode ajudar na decisão de usar ou não uma aplicação num dado momento.

As operações que o Serviço de Repositório de Aplicações disponibiliza aos seus clientes são as seguintes:

- listagem das aplicações disponíveis – devolve a lista contendo o nome de todas as aplicações disponíveis;
- listagem de metadados – devolve a informação relativa às aplicações disponíveis;
- obtenção da aplicação – permite fazer o carregamento de uma dada aplicação.

A disponibilização na interface de duas operações de listagem de aplicações permite que se possa optar, em tempo de implementação do serviço, por qualquer um dos dois métodos de construção da lista de aplicações descritos anteriormente. Aliás, será perfeitamente normal que se comece por uma abordagem mais simples nas primeiras versões de um dado programa e que se vá melhorando nas versões subsequentes. Com a evolução das versões, podem subsistir os dois métodos e usar-se um ou outro consoante o tipo de cliente que está a aceder ao serviço, já que podem existir vários tipos de cliente do sistema, quer seja porque os respectivos utilizadores não fizeram o carregamento de versões mais recentes, quer pelo facto de existirem diferentes versões para diferentes plataformas, sendo umas versões mais evoluídas do que outras.

A informação que se propõe que seja guardada no directório de aplicações é a seguinte:

- nome da aplicação;
- versão;
- tipo de aplicação – linguagem de programação usada na sua construção;
- mecanismo de controlo de concorrência – identifica qual o mecanismo de controlo de concorrência que a aplicação pode usar; serve para o serviço de controlo de concorrência verificar se uma aplicação que está a requerer

um dado serviço está autorizada a fazê-lo (este assunto é abordado na secção 4.9);

- breve descrição da aplicação, nomeadamente das suas facilidades cooperativas;
- metadados – informação acerca da aplicação, de modo a facilitar a sua procura;
- localização da aplicação.

A utilização do Serviço de Repositório de Aplicações foi já esquematizada quando se apresentou o diagrama de actividade do cliente do sistema, através da Figura 4-3.

4.7. Serviço de Repositório de Informação

O Serviço de Repositório de Informação permite aos utilizadores acederem aos ficheiros que contêm a informação a ser trabalhada pelas aplicações cooperativas. Para tal, este serviço permite ver a lista de ficheiros de informação disponíveis e permite seleccionar e carregar os ficheiros que se desejar.

À semelhança do que já foi discutido no Serviço de Repositório de Aplicações, também aqui se pode colocar o problema das versões dos ficheiros, uma vez que, potencialmente, estes serão acedidos e modificados por diversos utilizadores cooperativos, no mesmo instante ou em instantes distintos. Contudo, neste caso pode não interessar utilizar a última versão, mas sim uma versão anterior ou até a versão original, prevendo-se, assim, que sejam guardadas no sistema diversas versões do mesmo ficheiro. Desta forma, deve existir um mecanismo que as diferencie, seja através da forma como se constroem os nomes dos ficheiros, de modo a ser possível olhar para o nome e identificar a sua versão, ou recorrendo, mais uma vez, à utilização de metadados. As vantagens e desvantagens de cada uma das abordagens são semelhantes às já discutidas para o caso da lista de aplicações disponíveis. Poderá considerar-se ainda uma terceira hipótese, a de não efectuar qualquer controlo de versões. Neste caso, terá de haver um mecanismo que assegure que os ficheiros só sejam modificados

definitivamente por utilizadores privilegiados, podendo os outros utilizadores fazer apenas alterações locais. A escolha do mecanismo de controlo de versões de ficheiros depende do tipo de aplicações que o sistema suporta e das opções tomadas em tempo de implementação.

Na Figura 4-9 podem-se ver as ligações existentes entre o serviço de repositório de informação e as componentes do sistema com as quais interage directamente. A figura é, na realidade, muito semelhante à do repositório de aplicações, diferindo no serviço e na ligação com as aplicações e não com o cliente, bem como nas designações do directório e do armazenamento.

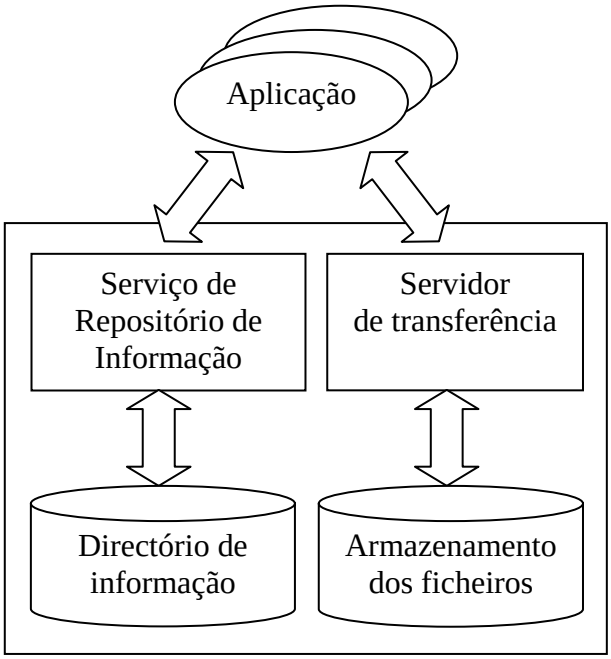

Figura 4-9 – Ligações do Serviço de Repositório de Informação

O servidor de transferência é utilizado pelas aplicações para proceder ao carregamento dos ficheiros que escolheu na lista que lhe foi apresentada. Outra função do servidor de transferência é receber das aplicações os ficheiros que estas produziram no decorrer do seu trabalho.

O armazenamento de ficheiros guarda todos os ficheiros de informação disponíveis para carregamento pelas aplicações, para serem manipulados nas actividades cooperativas. Tal como no caso do serviço de repositório de aplicações, também o serviço de repositório de ficheiros precisa de construir a lista de ficheiros a

que as aplicações têm acesso. Essa lista pode ser construída das mesmas maneiras que foram discutidas para o caso do repositório de aplicações, usando-se, eventualmente, informação relativa à detenção de controlo de acesso por parte de uma aplicação (*locks* ou testemunhos). Neste caso, a informação a ser guardada no directório de informação é a seguinte:

- nome do ficheiro;
- versão;
- tipo de informação;
- mecanismo de controlo de concorrência;
- breve descrição do conteúdo do ficheiro;
- metadados;
- localização do ficheiro;
- autor – permite saber quem colocou o recurso informativo no repositório.

As operações que o Serviço de Repositório de Informação disponibiliza às aplicações são as seguintes:

- listagem dos ficheiros disponíveis – devolve a lista contendo o nome de todas os ficheiros disponíveis;
- listagem de metadados – devolve toda a informação relativa aos ficheiros disponíveis;
- obtenção do ficheiro – permite à aplicação fazer o carregamento de um determinado ficheiro.

A disponibilização na interface de duas operações de listagem de ficheiros tem as mesmas razões que já foram apontadas para o caso do serviço de repositório de aplicações.

A Figura 4-10 mostra um diagrama de actividade que ilustra resumidamente a forma como as aplicações utilizam o Serviço de Repositório de Informação. Nele pode-se ver que, logo após iniciar a sua execução, a aplicação cria uma *proxy* do serviço de repositório de informação e invoca um método que lhe devolve a lista de ficheiros

disponíveis, eventualmente acompanhada de metadados. Nesta lista, o utilizador escolhe os ficheiros com os quais pretende trabalhar e inicia o carregamento de cada uma deles, caso não esteja disponível no disco local ou, eventualmente, apesar de já estar disponível uma versão local, esta não seja a versão desejada.

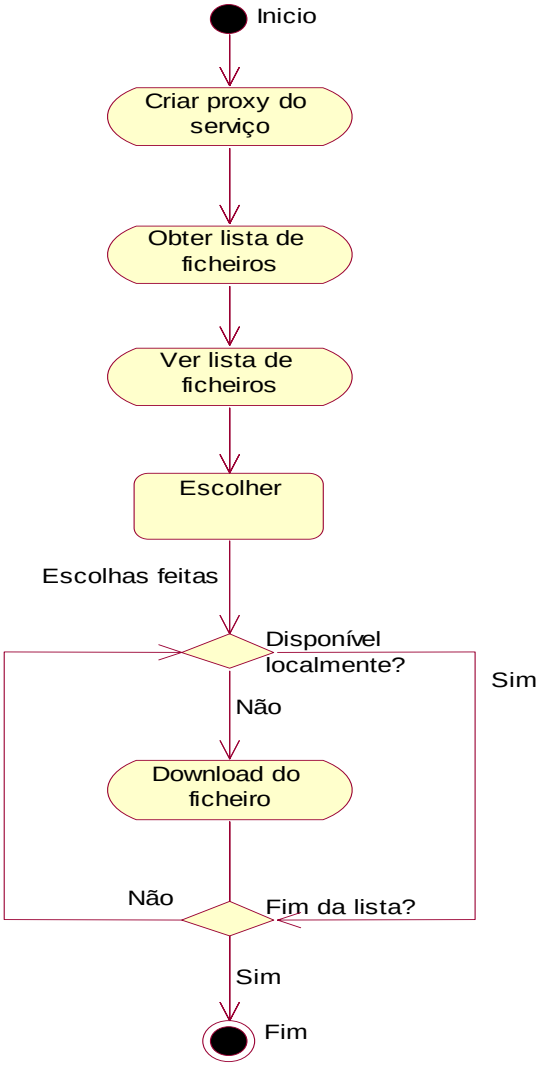

Figura 4-10 – Diagrama de actividade da utilização do repositório de informação

Olhando para os serviços de repositório de aplicações e de repositório de informação, verifica-se que eles possuem diversos pontos comuns, nomeadamente:

- interface muito parecida: a única diferença está no facto de um serviço lidar com ficheiros que contêm aplicações e o outro lidar com ficheiros contendo informação textual ou audiovisual;
- os metadados necessários são também muito semelhantes, residindo a principal diferença, mais uma vez, no facto de se lidar com aplicações ou com ficheiros de informação.

Desta forma, em tempo de implementação, pode ser vantajoso integrar os dois serviços num só, simplificando a arquitectura global do sistema.

4.8. Serviço de Notificação de Eventos

A notificação de eventos é um aspecto fundamental, para uma grande parte das aplicações cooperativas, pela possibilidade que oferece de as acções de um determinado utilizador poderem ser reflectidas em todos os outros utilizadores que estejam interessados em tomar conhecimento dessas acções. Esta característica é particularmente importante no caso das aplicações em que existem contextos partilhados, tais como áreas de desenho, editores de texto ou de outros media, botões ou apontadores.

A Figura 4-11 ilustra simplificadamente a arquitectura proposta para o Serviço de Notificação de Eventos. Quando uma aplicação tem interesse em receber notificações de um determinado tipo de eventos, regista esse interesse enviando uma mensagem a uma instância do serviço de notificação de eventos, a qual, por sua vez, propaga o registo ao sistema de notificação de eventos. Quando os eventos ocorrem, o sistema de notificação de eventos envia a respectiva mensagem de notificação a todas as instâncias do serviço registadas. Cada uma destas, por sua vez, encarrega-se de entregar a notificação à aplicação que a instanciou. Para que tal seja possível, as notificações são enviadas em mensagens assíncronas, pois se assim não fosse, só seria possível a propagação dos eventos se as aplicações fossem elas próprias serviços com métodos invocáveis.

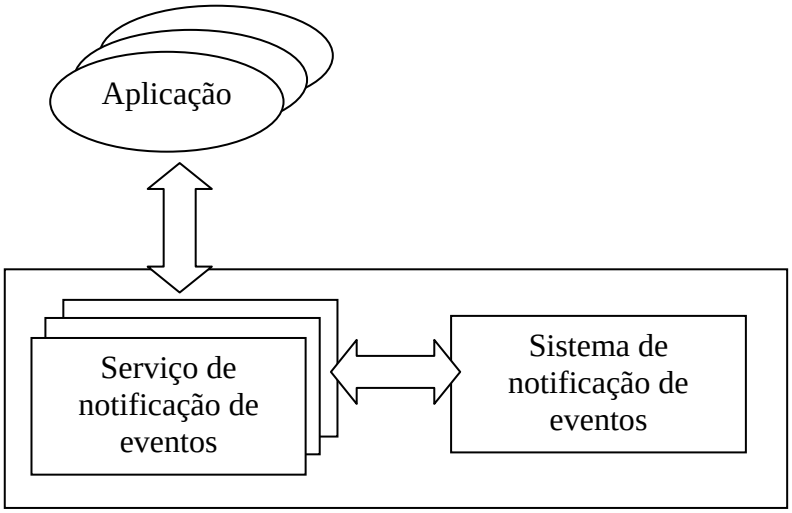

Figura 4-11 – Arquitectura do Serviço de Notificação de Eventos

A interface do serviço de notificação de eventos disponibiliza às aplicações as seguintes operações:

- registo de interesse em evento – permite às aplicações registarem o seu interesse em serem notificadas acerca dos eventos de um determinado tipo que ocorram nas outras aplicações com as quais cooperam;
- remoção de interesse em eventos – permite às aplicações retirarem o seu interesse em serem notificadas acerca de determinado evento, porque vão cessar a sua actividade, ou pelo menos porque vão cessar a cooperação;
- notificação de evento – permite às aplicações notificarem a ocorrência dos seus próprio eventos, de modo a que estes possam ser propagados às outras aplicações cooperantes.

A Figura 4-12 mostra o diagrama de actividade da utilização do serviço de eventos por parte das aplicações. Após as diversas inicializações que a aplicação faz para dar início à sua actividade (objectos da interface gráfica, variáveis, etc.), cria uma *proxy* do Serviço de Notificação de Eventos, para que possa iniciar a usa utilização.

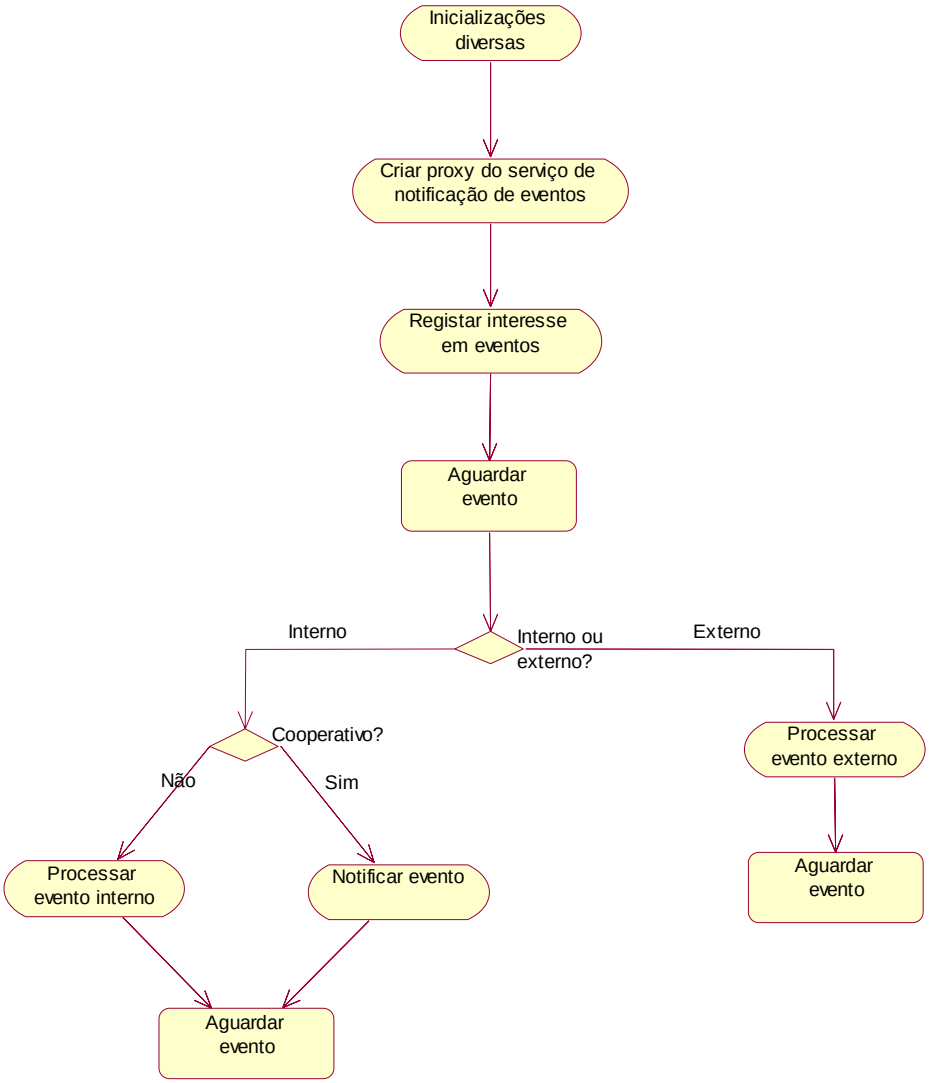

Figura 4-12 – Diagrama de actividade da utilização dos eventos nas aplicações

Na aplicação há que distinguir eventos internos e eventos externos. Os eventos internos são os produzidos na própria aplicação e podem apenas ter reflexo local ou podem ser eventos públicos, que devem ser propagados a todas as aplicações que manifestaram interesse em serem notificadas acerca da sua ocorrência. No caso dos eventos serem internos, são processados segundo os métodos normais – por exemplo, em Java as acções do rato têm métodos próprios para captarem a ocorrência de acções relacionadas com essa actividade e agirem de acordo com o que o programador especificou. Caso esses eventos tenham reflexo externo, é também enviada uma

mensagem de notificação da sua ocorrência ao serviço de notificação de eventos, invocando a operação que este serviço disponibiliza para o efeito.

Quando a aplicação recebe uma mensagem do Serviço de Notificação de Eventos, notificando-a da ocorrência de um evento externo de determinado tipo, a aplicação processa o evento, agindo de acordo com o tipo de evento notificado.

A aplicação está permanentemente à espera da ocorrência de eventos, quer desencadeados internamente, quer externamente. Este facto é ilustrado na Figura 4-12 pelo estado "Aguardar evento", ao qual todos os caminhos do diagrama vão dar. Há, contudo, uma situação em que um dos eventos diz respeito ao encerramento da aplicação, o qual não se encontra representado no diagrama por uma questão de clareza de leitura.

4.9. Serviço de Controlo de Concorrência

Como foi visto no Capítulo 3, o controlo de concorrência pode ter uma abordagem optimista ou pessimista. Na abordagem optimista assume-se a não existência de conflitos no acesso aos recursos informativos, não havendo garantia da consistência dos dados, podendo até permitir-se a manipulação de dados inconsistentes. Numa abordagem pessimista admite-se que os conflitos podem ocorrer e ser de grande gravidade, controlando-se o acesso à informação partilhada, de uma forma centralizada ou descentralizada. Na primeira, existe uma entidade central que efectua o controlo, fazendo, por exemplo, a passagem de testemunho segundo uma ordem pré-estabelecida (*token passing*) entre as várias entidades envolvidas. Na segunda, a responsabilidade pelo controlo do acesso aos recursos informativos é partilhada pelos membros do grupo. Neste caso, podem existir mecanismos de votação, passagem de testemunho sem ordem pré-estabelecida (*floor-passing*) ou *locks*.

Os mecanismos de controlo de concorrência fazem sentido quando as alterações a efectuar vão sendo feitas no ficheiro original. No modelo de serviços cooperativos proposto nesta tese, é contemplada a hipótese de enviar uma cópia do ficheiro para o computador do utilizador, pelo que podemos ter inúmeras cópias do mesmo ficheiro,

potencialmente com diferenças substanciais entre elas. Neste caso existem duas soluções:

- adoptar um sistema de versões – de cada vez que um utilizador quer transferir o resultado do seu trabalho para o armazenamento de grupo, o ficheiro correspondente é armazenado em separado;
- utilizar a notificação de eventos – todas as alterações de todos os utilizadores cooperantes são reflectidas por intermédio do Serviço de Notificação de Eventos.

No modelo proposto para a criação de serviços cooperativos, o mecanismo de controlo de concorrência varia consoante o tipo de aplicações cooperativas em questão. Assim, para aplicações de edição de grupo, procede-se à transferência de cópias dos ficheiros para cada computador e utiliza-se o sistema de versões e/ou a notificação de eventos, consoante o tipo de aplicação. Noutros casos, adopta-se um mecanismo de *floor-passing*, podendo haver um utilizador com a responsabilidade de atribuir a vez a cada um dos outros utilizadores ou essa atribuição ser acordada, em cada instante, entre os utilizadores, assumindo-se um ambiente de cordialidade e confiança entre todos. Em casos muito específicos, pode ainda não haver qualquer controlo, como é o caso de um quadro partilhado em que cada utilizador pode escrever ou desenhar livremente, sendo as suas acções reflectidas pelo Serviço de Notificação de Eventos.

Desta forma, o Serviço de Controlo de Concorrência possui na sua interface as seguintes operações, de forma a conseguir contemplar o maior número possível de políticas de controlo de concorrência:

- obtenção de *lock* – permite às aplicações assegurarem o acesso exclusivo aos recursos;
- libertação de *lock* – permite às aplicações libertarem os recursos previamente bloqueados; exige especial cuidado com a situação de uma aplicação deter um recurso indefinidamente, pelo que terá que haver um tempo limite de bloqueio do recurso;

- obtenção de testemunho – permite a atribuição da vez no controlo de recursos, segundo os mecanismos de *token passing* e *floor-passing*; este mecanismo pode exigir a existência de uma entidade controladora da sequência de passagem de testemunho;
- libertação de testemunho – liberta o testemunho para que possa ser passado a outra aplicação;
- pedido de versão – usado pelas aplicações de edição de grupo para requererem a atribuição de um número de versão aos ficheiros que pretendem guardar no armazenamento de grupo;
- início de votação – pedido de início do processo de votação acerca de um determinado recurso, fazendo as inicializações necessárias à recolha dos votos e sua contagem; este pedido deve ser feito pela entidade responsável pelo processo de votação;
- envio de voto – permite às aplicações submeterem o seu voto;
- resultado da votação – permite recolher o resultado da votação.

Cada invocação de uma destas operações inclui informação relativa ao tipo de aplicação que a está a efectuar. Esta informação é usada pelo Serviço de Controlo de Concorrência para verificar no directório de aplicações se é permitido a este tipo de aplicação usar o mecanismo de controlo de concorrência que está a requerer.

A Figura 4-13 mostra a arquitectura do Serviço de Controlo de Concorrência, bem como as suas ligações com as outras componentes do sistema. Nela faz-se uma subdivisão do serviço em módulos correspondentes às políticas de controlo de concorrência que o serviço implementa, as quais se encontram também distinguidas por cores, tais como os diferentes tipos de aplicações que as utilizam.

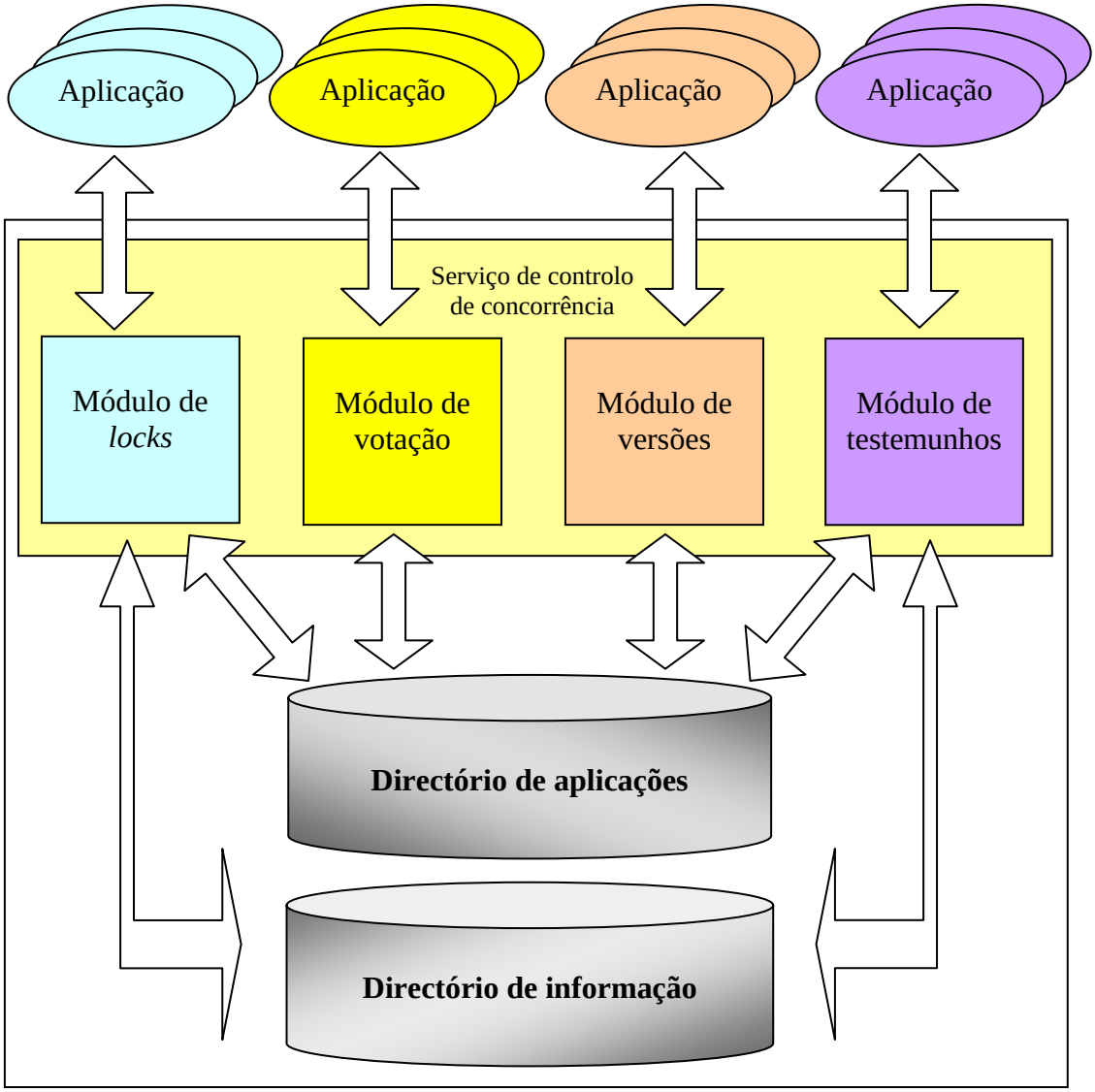

Figura 4-13 – Arquitectura do Serviço de Controlo de Concorrência

4.10. Cenários de aplicação do modelo

O modelo de criação de serviços cooperativos proposto nesta tese adapta-se a uma grande variedade de aplicações de trabalho cooperativo suportado por computador. De facto, a adopção de soluções arquitecturais suficientemente genéricas possibilita ao modelo apresentar uma versatilidade que lhe permite contemplar a generalidade das categorias de aplicações cooperativas. De entre as características do modelo, destacam-se as seguintes, no que diz respeito à referida generalidade:

- as características dos *web services*, no que concerne à interoperabilidade e à capacidade de aproveitamento de sistemas legados, conferem-lhe uma grande flexibilidade relativamente aos tipos de aplicações que os podem usar;
- o Serviço de Notificação de Eventos é genérico, permitindo a criação de diversas sessões cooperativas e a propagação de variados tipos de eventos;
- o Serviço de Controlo de Concorrência contempla diversos mecanismos de controlo de concorrência, adaptando-se a um grande número de situações.

Desta forma, passa-se a analisar a adequação do modelo a diversas categorias de aplicações cooperativas.

Sistemas de mensagens

Identificando-se o tipo de evento como sendo uma mensagem de texto, o texto da mensagem que se deseja enviar constitui o valor do evento, sendo passado como parâmetro da invocação de notificação. No que diz respeito ao Serviço de Controlo de Concorrência, este não coloca quaisquer problemas, pois este tipo de aplicações não necessita de controlar o acesso a recursos informativos partilhados. Desta forma, o modelo proposto adequa-se a este tipo de aplicação cooperativa, a qual, contudo, é habitualmente usada como aplicação de suporte à comunicação entre elementos de um grupo de trabalho envolvido noutras actividades cooperativas [127].

Videoconferência

Neste caso, a notificação de eventos apenas poderá servir para avisar os intervenientes acerca da entrada e saída de utilizadores. Trata-se, na realidade, de um tipo de aplicação que serve de suporte a outras aplicações cooperativas, apenas com o intuito de proporcionar a comunicação entre os elementos do grupo de trabalho, não

necessitando de aceder a recursos de informação partilhados [59, 128-131]. Desta forma, a funcionalidade de videoconferência será implementada por uma aplicação e não por um serviço, dada as suas características específicas que exigem o melhor desempenho possível, recorrendo a protocolos de comunicação vocacionados para o efeito. O modelo proposto continua a contemplar, contudo, a sua disponibilização para carregamento através do Serviço de Repositório de Aplicações.

Espaços de informação partilhada

Os aspectos mais importantes no caso dos espaços de informação partilhada [74] são os relacionados com o controlo de concorrência e com a organização da informação. No que diz respeito ao primeiro aspecto, o Serviço de Controlo de Concorrência disponibiliza diversos mecanismos, pelo que fica contemplado um grande número de situações. Em relação à organização da informação, é importante a sua estruturação hierárquica. O armazenamento da informação com metadados permite que existam aplicações que usem esses metadados para estruturar a informação.

Sistemas de coordenação de actividades (*Workflow*)

Neste tipo de aplicações [80-82, 124, 132-135] não existe cooperação em tempo real, mas sim a atribuição coordenada de tarefas a diferentes intervenientes segundo uma ordem predefinida. A execução das tarefas é maioritariamente sequencial, mas podem existir actividades paralelas, nas quais os utilizadores não interagem directamente. Os mecanismos de controlo de concorrência do tipo *token passing* adequam-se bem à situação de execução sequencial de tarefas e no caso da execução paralela podem ser utilizados os mecanismos de versões e de *locks*, dependendo da natureza das actividades. Outro aspecto importante nas aplicações de *workflow* é a monitorização das actividades, que pode ser implementada através do sistema de eventos. O modelo proposto satisfaz todos estes requisitos.

Salas de reunião electrónicas

No que diz respeito aos sistemas de reuniões electrónicas [85-87, 89, 90, 136], o Serviço de Notificação de Eventos fornece os mecanismos necessários à actualização da informação presente em quadros virtuais e outros recursos informativos partilhados. Permite ainda a utilização de outros dispositivos partilhados, como é o caso dos teleapontadores, ou a propagação de acções em botões. O Serviço de Controlo de Concorrência fornece os meios para controlar o acesso concorrente à informação partilhada, seguindo a política mais adequada. No caso de um quadro virtual partilhado o Serviço de Notificação de Eventos é suficiente para reflectir as acções dos diversos utilizadores. Em documentos partilhados dispomos de mecanismos de controlo de versões e de passagem de testemunho.

Editores de grupo

O caso dos editores de grupo [94-98, 137] já foi referido quando se apresentou o Serviço de Controlo de Concorrência. De facto, neste caso podemos optar por um sistema de versões dos documentos ou media partilhados, havendo cópias locais dos respectivos ficheiros. Para além disso, continua a ser possível aos utilizadores terem conhecimento das alterações efectuadas pelos outros, por intermédio do Serviço de Notificação de Eventos. Também se pode adoptar uma solução mista, em que os utilizadores possuem cópias locais dos ficheiros nas quais efectuam alterações, sendo estas propagadas aos outros utilizadores e a um gestor do recurso, ao qual cabe decidir quais as alterações que serão efectivadas na versão final. Pode ainda optar-se por validar as alterações através de um mecanismo de votação.

Agentes cooperantes

Os agentes cooperantes são usados tipicamente na procura de informação e na resolução distribuída de tarefas.

A procura de informação pode ser feita recorrendo a agentes móveis [138, 139] e envolve a localização das fontes, a sua pesquisa, a recolha dos resultados e a extracção de informação útil. Um único agente pode ser responsável por todas estas tarefas, existindo vários agentes pesquisando em diferentes fontes de informação. A forma de cooperação consiste na actividade em si e no envio dos resultados para uma entidade gestora. A notificação de eventos é suficiente para assegurar a transmissão dos resultados, de instruções aos diversos agentes ou para, eventualmente, os notificar acerca da necessidade de concluir antecipadamente a sua execução. O problema do controlo de concorrência apenas se poderá colocar do lado da aplicação gestora da actividade dos agentes e será fortemente dependente das opções que forem tomadas para a sua implementação. No caso dos vários agentes possuírem tarefas distintas no processo de procura de informação, a prossecução de uma tarefa está dependente da conclusão da anterior, num processo semelhante ao de um sistema de *workflow*, para o qual já foi discutida a adequação do modelo.

Na resolução distribuída de tarefas, para além de cenários semelhantes ao descrito para a procura de informação com atribuição de tarefas distintas a vários agentes, há também o caso da marcação distribuída de reuniões, já apresentado no Capítulo 3. Neste caso, o Serviço de Controlo de Concorrência fornece os meios de votação que possam ser necessários à negociação de um acordo.

Ensino assistido por computador

Na categoria de sistemas de ensino assistido por computador encontram-se as mais tradicionais soluções baseadas em CD-ROM ou em simples páginas *web*, mas também aplicações mais sofisticadas, como a aula electrónica ou a aula virtual [108, 109, 113, 114, 117, 128, 140-142]. Nos dois últimos casos, o computador é usado como ferramenta de apoio didáctico, possibilitando o acesso a documentação variada e fornecendo um quadro electrónico partilhado, para facilitar a troca de informação entre os intervenientes. No caso da aula virtual acrescem ainda as ferramentas de comunicação entre todos os intervenientes, para as quais foi já discutida a adequação do modelo proposto, tal como acontece com o quadro electrónico partilhado. No acesso à

documentação podem ser implementadas diversas políticas de controlo de acesso concorrente à informação, disponíveis no Serviço de Controlo de Concorrência. Em princípio, esse controlo será exercido apenas sobre a escrita nos documentos, a qual só deverá ser permitida ao professor, sendo necessário proceder ao bloqueio permanente do recurso.

Ambiente cooperativo virtual

Nesta categoria de aplicações cooperativas inserem-se as aplicações de representação tridimensional de objectos ou pessoas [119, 121, 143-145] e os sistemas de imersão [120]. No primeiro caso, trata-se de simular a presença de objectos ou pessoas, segundo uma representação poligonal simplificada, para a qual apenas é necessário transmitir eventos ocorridos na posição das diversas formas geométricas da representação. No segundo caso, pretende-se criar no utilizador a ilusão de se encontrar num ambiente envolvente remoto. Nesta situação, o mais complexo são os recursos físicos necessários à criação de tal ilusão (monitores e sua disposição).

****

Verifica-se, assim, haver uma grande variedade de aplicações cooperativas que podem ser implementadas usando o modelo proposto nesta Tese. No próximo capítulo será apresentado um ambiente para a criação de aplicações cooperativas baseado no modelo aqui proposto e um protótipo de uma aplicação cooperativa para o testar.

Capítulo 5

Um Ambiente Cooperativo baseado no Modelo de Criação de Serviços Cooperativos

---

Este capítulo apresenta um ambiente de suporte a aplicações cooperativas baseado no modelo de criação de serviços cooperativos descrito no Capítulo 4. O ambiente apresentado implementa uma aplicação simplificada de edição cooperativa de vídeo e os serviços mais importantes para a validação do modelo proposto. Na descrição do mesmo são discutidas as opções de implementação disponíveis e as razões para as escolhas efectuadas. Por fim, faz-se uma breve análise do desempenho do sistema.

5.1. Introdução

O modelo de criação de serviços cooperativos apresentado no Capítulo 4 procura ser o mais genérico possível, de forma a poder adaptar-se a um leque variado de actividades cooperativas suportadas por computador. Contudo, a validação do modelo para diversos tipos de aplicações cooperativas exige a implementação de uma grande variedade de aplicações relativamente complexas. Assim, escolheu-se uma aplicação multimédia cooperativa que permitisse validar, em tempo útil, o essencial do modelo proposto. Os resultados obtidos para esta aplicação permitem perspectivar, com algum optimismo, a validade da utilização do modelo noutros cenários cooperativos.

Na implementação do ambiente baseado no modelo de serviços cooperativos, efectuaram-se alguns ajustes na estrutura dos serviços, resultantes de condicionalismos e opções de implementação, mas que, contudo, não afectam a sua validade para outras aplicações cooperativas. Nestes ajustes, destacam-se os seguintes aspectos:

- o Directório de Utilizadores foi integrado no Serviço de Notificação de Eventos, pois todo o processo de registo e manutenção de informação de actividade cooperativa está intimamente ligado ao sistema de propagação de eventos e a informação associada não é persistente, não necessitando de ser armazenada em base de dados. Porém, continua a ser válida a separação dos dois serviços em termos de modelo, visto que se podem tomar opções distintas noutras implementações desses serviços. Por exemplo, pode-se optar por tornar persistente a informação relativa à actividade cooperativa, sendo registados em base de dados os utilizadores que estão activos e as actividades em que se encontram envolvidos;
- o Serviço de Controlo de Concorrência não se encontra implementado num *web service* específico, pois o mecanismo de controlo de concorrência escolhido para a aplicação cooperativa, baseado em versões, é controlado pelos serviços de repositório, os quais efectuam o armazenamento dos recursos nas bases de dados. Refira-se, no entanto, que o sistema de gestão de bases de dados de objectos que foi escolhido para implementar as bases de dados deste ambiente cooperativo, permite a

definição de *locks* nos objectos, garantindo assim a consistência desses objectos. Para além disso, a API de distribuição de eventos, *Java Shared Data Toolkit* (JSDT), que será introduzida na secção 5.2, permite definir *Tokens*, que possibilitam a construção de esquemas de passagem de testemunho, que também podem ser usados para implementar mecanismos de controlo de concorrência;

- as ferramentas de monitorização do sistema também não foram implementadas, pois não são relevantes no contexto da validação que se pretende para o modelo proposto.

A linguagem de programação escolhida para implementar as várias componentes do ambiente cooperativo que é apresentado neste capítulo foi a linguagem **Java**. As razões dessa escolha estão relacionadas com o facto de ser uma linguagem orientada a objectos com boas capacidades sintácticas e semânticas e que também possui várias APIs úteis para o tipo de sistemas que se pretendem implementar, nomeadamente APIs de notificação de eventos, de acesso e manipulação de bases de dados orientadas a objectos e de manipulação de informação multimédia.

Actualmente, a indústria de *software* disponibiliza vários ambientes de desenvolvimento de *web services* em Java, se bem que muitos não tenham atingido ainda a maturidade necessária para possuirem uma comunidade de utilizadores significativa. De entre os ambientes de desenvolvimento de *web services* disponíveis no mercado, onde ainda subsistem muitos produtos de utilização gratuita, destacam-se alguns mais difundidos: Systinet WASP Server for Java [146]; IONA Artix [147]; BEA WebLogic [52]; FusionWare Integration Server [148]; e Sun Java WSDP (*Web Services Developer Pack*) [149]. Este último não possui nenhuma aplicação de desenvolvimento ou projecto específica, apenas disponibilizando uma API Java.

A escolha do ambiente de desenvolvimento de *web services* recaiu sobre o **WASP**, por ser um sistema que apresenta já alguma maturidade, estar razoavelmente documentado, ter uma vasta comunidade de utilizadores, ser gratuito e poder ser integrado nos principais ambientes de desenvolvimento de programas Java (vulgo IDEs), como NetBeans [150] e Eclipse [151]. A versão do WASP utilizada possui um

plugin para integrar o desenvolvimento de *web services* no IDE **NetBeans**, o qual foi escolhido por uma questão de familiarização com a ferramenta. Este ambiente de desenvolvimento de *web services* permite codificar as interfaces dos serviços como aplicações Java comuns e possui ferramentas que permitem gerar automaticamente os ficheiros WSDL contendo as descrições dos serviços e os ficheiros de adaptação para os clientes (*proxies*) a partir das interfaces. Para além disso, possui também o ambiente de execução (*runtime*) necessário à activação dos serviços, à sua publicação e à transformação das invocações de métodos Java em mensagens SOAP e à correspondente transformação inversa no destinatário.

Para o armazenamento da informação optou-se por bases de dados de objectos. A discussão das vantagens e desvantagens dos modelos de bases de dados relacionais e de objectos não conduz a consensos, verificando-se que existe uma grande divisão de opiniões, as quais estão fortemente relacionadas com as práticas habituais dos programadores e também com os ambientes de desenvolvimento existentes [152-154]. Desta forma, quando as aplicações que vão usar as bases de dados se baseiam num modelo de objectos e a estrutura da informação pode ser dinâmica, parece ser vantajosa a utilização de sistemas de bases de dados de objectos.

5.2. Arquitectura do sistema

A Figura 5-1 apresenta a arquitectura do ambiente cooperativo resultante da utilização do modelo de criação de serviços cooperativos. O diagrama de blocos é muito semelhante ao apresentado no Capítulo 4, faltando-lhe os serviços de Directório de Utilizadores e de Controlo de Concorrência, pelas razões já apresentadas.

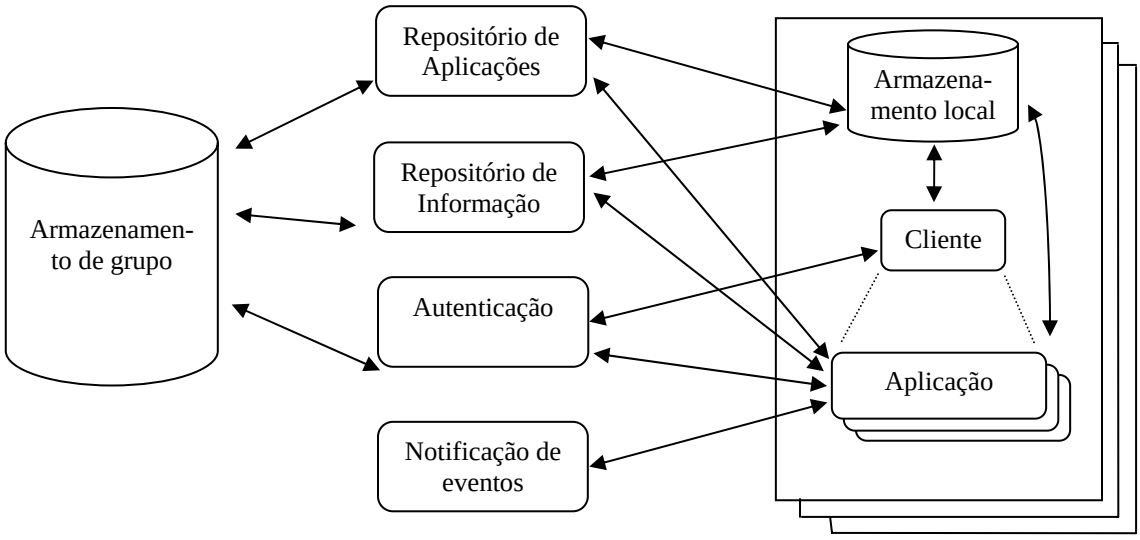

Figura 5-1 – Arquitectura do ambiente baseado no modelo de criação de serviços cooperativos

Para um utilizador poder usar as funcionalidades do ambiente cooperativo tem que primeiro ser autenticado, através do Serviço de Autenticação. Após ser autenticado, o utilizador usa uma aplicação cooperativa que já possua no seu computador (edição cooperativa de vídeo, por exemplo) ou usa uma ferramenta que permite ver a lista de aplicações disponíveis no Repositório de Aplicações e carregar a que deseja utilizar. Quando o utilizador começa a usar uma aplicação cooperativa, fá-lo em modo isolado, ou seja, não estabelece nenhuma sessão cooperativa com outros utilizadores. Contudo, fica logo registado no Serviço de Notificação de Eventos (que integra também as funcionalidades do directório de utilizadores, como já foi referido na introdução deste capítulo) como estando activo, podendo ser convidado por outros utilizadores a participar numa sessão de trabalho cooperativo.

Um utilizador que deseja iniciar uma sessão cooperativa com outro utilizador, consulta no Serviço de Notificação de Eventos a lista de utilizadores da mesma aplicação que se encontram activos no momento. Nesta lista pode seleccionar um utilizador e convidá-lo para uma sessão cooperativa. Caso esse utilizador aceite o convite, ambos se registam numa sessão cooperativa no Serviço de Notificação de Eventos. A estes utilizadores podem-se juntar outros, sob convite de um já registado na mesma sessão e qualquer um deles pode abandonar a sessão quando desejar. Os eventos cooperativos produzidos durante uma sessão são transmitidos a todos os utilizadores que se encontrem registados nessa sessão no momento em que esses eventos ocorrem,

sendo essa transmissão da responsabilidade do Serviço de Notificação de Eventos. Quando um utilizador encerra uma aplicação cooperativa é retirado de todas as sessões cooperativas em que eventualmente se encontre registado e das quais não tenha ainda saído, sendo igualmente retirado o seu registo como utilizador activo no Serviço de Notificação de Eventos.

Cada utilizador que queira usar um *web service* deve dispor de uma aplicação cliente desse serviço, a qual é parcialmente gerada pela infra-estrutura WASP e depois completada pelo programador com instruções específicas de cada caso.

O componente do sistema que na arquitectura assume a designação de Cliente é implementado no ambiente cooperativo como o cliente do Serviço de Autenticação.

A Figura 5-2 mostra o diagrama de classes do sistema implementado para validar o modelo de serviços cooperativos. Nele podem-se ver as classes que implementam os serviços, as suas interfaces, as diversas aplicações que utilizam os serviços, as classes das bases de dados e as interacções entre todos os componentes.

Os clientes dos serviços não interagem directamente com estes, mas sim com as *proxies* que possuem do lado do cliente, representadas na Figura 5-2 pelos círculos azuis. Por exemplo, apesar da figura representar uma ligação directa entre o editor de vídeo (`EditorVideo`) e o serviço de eventos (`sEventos`), esta ligação é feita por intermédio da *proxy* do serviço de eventos (`SEventos`). Por uma questão de clareza do diagrama, não se encontra representada a ligação entre a aplicação de acesso ao Repositório de Aplicações (`listaAplics`) e o Serviço de Autenticação.

A entrada no sistema é feita através de uma aplicação cooperativa (`EditorVideo`, no ambiente cooperativo apresentado neste capítulo) ou da aplicação `listaAplics`, que se comporta como cliente do Serviço de Repositório de Aplicações (`repAplic`). Cada aplicação do sistema faz a validação do utilizador usando o Serviço de Autenticação (`controlador`). Caso o utilizador não possua ainda a aplicação que pretende usar, após a sua autenticação, utiliza o Repositório de Aplicações para escolher a aplicação e fazer o seu carregamento a partir da base de dados que serve de suporte ao repositório. Depois de carregar as aplicações, o cliente executa-as, utilizando para o efeito as potencialidades oferecidas pela linguagem Java, através de um *Class Loader*.

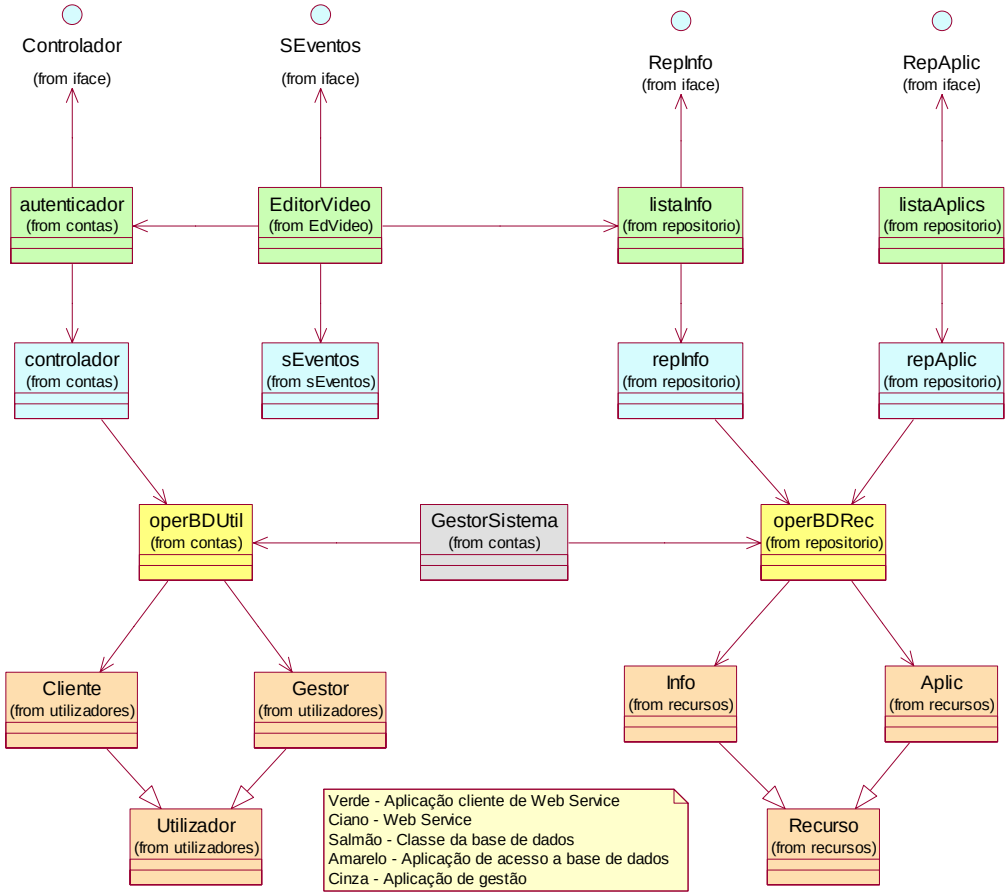

Figura 5-2 – Diagrama de classes do ambiente cooperativo

Neste ambiente cooperativo apenas se implementou uma aplicação cooperativa, a qual permite fazer **edição de vídeo** em formato digital de uma forma muito simplificada, definindo pontos de corte (*IN* e *OUT*) numa sequência de vídeo. A referida aplicação (`EditorVideo`) usa o serviço de Repositório de Informação, por intermédio da sua aplicação de acesso (`listaInfo`), para o utilizador escolher os ficheiros multimédia que deseja manipular e que não estejam armazenados localmente. O editor de vídeo usa também o Serviço de Notificação de Eventos (`sEventos`) para reflectir as suas actividades cooperativas e para ser notificado das actividades cooperativas dos seus parceiros de cooperação.

A aplicação de edição de vídeo usa uma API da linguagem Java, designada **JMF** (*Java Media Framework*) [155], que fornece meios para o processamento e a

visualização de ficheiros multimédia em diversos formatos. A sua utilização é explicada mais pormenorizadamente na secção 5.9.

O Serviço de Notificação de Eventos recorre a uma API da linguagem Java específica para a propagação de eventos entre aplicações. Essa API, designada **JSDT** (*Java Shared Data Toolkit*) [156], possui um conjunto de classes e métodos que permitem às aplicações Java criar canais e sessões de propagação de eventos, aos quais outras aplicações se juntam para poderem receber essas notificações. A utilização das classes dessa API usadas neste ambiente cooperativo é explicada com mais detalhes na secção 5.8.

Os serviços de repositório e de autenticação necessitam de guardar informação de uma forma persistente, recorrendo para o efeito a duas bases de dados, uma de utilizadores (`BDU`) e outra de recursos (`BDR`). Ambas são bases de dados de objectos, como já foi referido na introdução deste capítulo, e seguem a API JDO (*Java Data Objects*) [157], disponibilizada pelo produto *FastObjects Trial Edition* [158]. A escolha deste produto, de entre os que foram encontrados no mercado, prendeu-se com o facto de ser uma base de dados de objectos e não uma ponte objectos-relacional, como acontece com muitos dos produtos publicitados como bases de dados JDO, para além do facto de dispor de versões gratuitas de avaliação.

5.3. Autenticação

A autenticação dos utilizadores é essencial para restringir o acesso ao sistema a pessoas autorizadas. O Serviço de Autenticação fornece uma interface que disponibiliza os meios necessários para que uma qualquer aplicação, escrita numa linguagem de programação que consiga aceder a *web services*, possa submeter, para validação, a identificação e a senha de um utilizador. Para tal, a interface do serviço de autenticação apenas precisa de disponibilizar um método ao qual é passada essa informação e que devolve o resultado da validação:

```
public boolean verificar(String nome, String senha)
```

Os parâmetros do método são duas *strings*, uma contendo a identificação do utilizador (*login*) e outra a respectiva senha (*password*), procedendo depois o serviço à sua validação, através da pesquisa em base de dados do nome do utilizador (`nome`) e da comparação da senha introduzida (`senha`) com a que se encontra armazenada. Se a comparação for bem sucedida, o método retorna `true`, ficando o utilizador habilitado a aceder aos recursos do sistema. A Figura 5-3 mostra a janela onde o utilizador insere os dados para proceder à sua autenticação.

Figura 5-3 – Janela de autenticação

Para que uma aplicação possa usar um *web service*, deve fazer algumas inicializações para preparar a interacção com o serviço. A seguir apresentam-se as instruções que preparam a interacção com o Serviço de Autenticação, sendo a última instrução já a invocação do método de autenticação do utilizador:

```
Controlador service;
String wsdlURI = "http://"+endIP+endAut;
String serviceURI = "http://"+endIP+endAut;
ServiceClient serviceClient = ServiceClient.create(wsdlURI);
serviceClient.setServiceURL(serviceURI);
service=(Controlador)serviceClient.createProxy(Controlador.class);
sucesso=service.verificar(this.nome,senha); .
```

Os dados dos utilizadores encontram-se armazenados numa base de dados (`BDU`), na qual são guardados objectos de duas classes: `Gestor` e `Cliente`. Ambas as classes derivam de uma classe `Utilizador`, a qual possui uma série de atributos comuns aos

dois tipos de utilizadores, sendo a diferenciação feita pelas subclasses. A Figura 5-4 mostra o diagrama de classes do Serviço de Autenticação, onde são visíveis as classes que implementam o serviço (`controlador` e `Controlador`), a aplicação cliente do serviço (`autenticador`), a aplicação que executa as operações na base de dados (`operBDUtil`), as classes dos objectos armazenados (`Utilizador`, `Gestor` e `Cliente`) e a aplicação de gestão do sistema (`GestorSistema`).

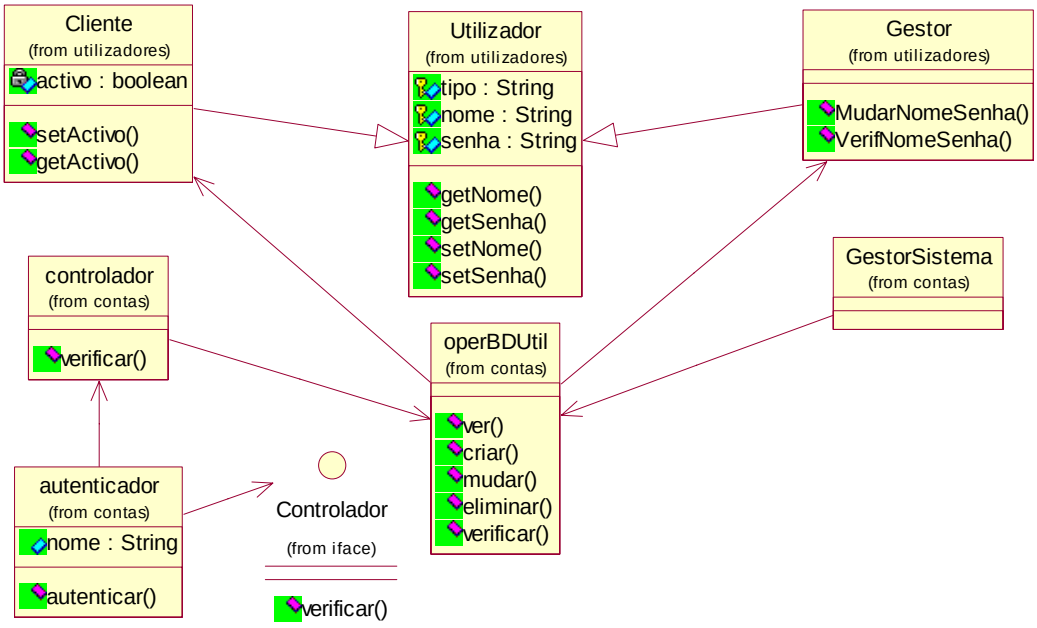

Figura 5-4 – Diagrama de classes do Serviço de Autenticação

5.4. Base de dados de utilizadores

A base de dados de utilizadores armazena objectos de duas classes, `Cliente` e `Gestor`, as quais são subclasses da classe `Utilizador`, herdando todos os seus atributos e métodos, acrescentando apenas os elementos que são específicos dos utilizadores comuns ou dos gestores. Esta hierarquia está representada na Figura 5-5.

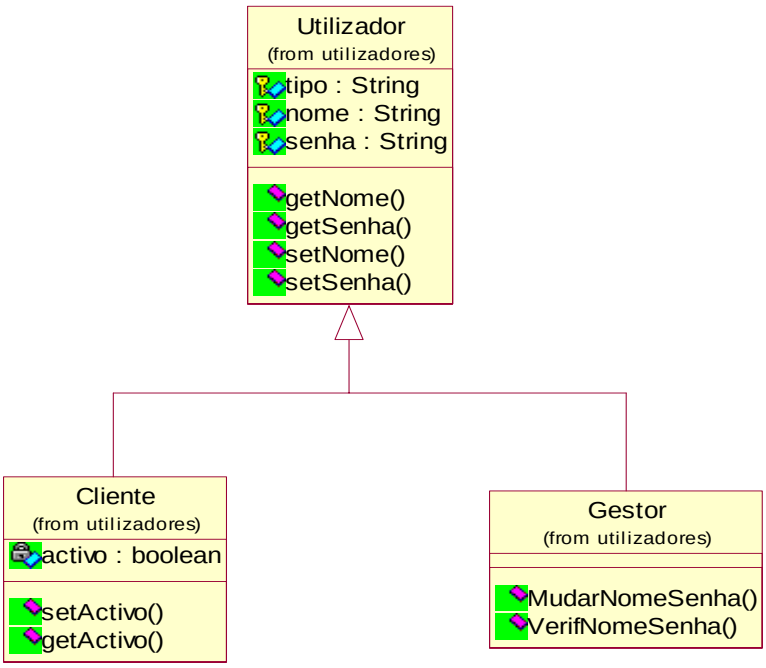

Figura 5-5 – Classes da base de dados de utilizadores

A classe `Utilizador` possui três atributos: `tipo`, `nome` e `senha`. O atributo `tipo`, identifica se o utilizador é gestor ou utilizador comum, podendo ser usado para fins de catalogação dos utilizadores. Tal funcionalidade não é utilizada actualmente, pois é possível definir nas bases de dados colecções de objectos (*Index*) de acordo com a sua classe e, neste caso, está definido um *index* para objectos `Cliente` (`ClienteNomeIdx`) e outro para objectos `Gestor` (`GestorNomeIdx`). No entanto, manteve-se o atributo `tipo`, para o caso de vir a ser necessário fazer uma subdivisão dos clientes em várias categorias.

A estrutura de uma base de dados que siga a especificação JDO é definida num ficheiro XML, onde se identifica a base de dados, o esquema que ela segue, as classes que armazena e os *índexes* que se querem criar, entre outros elementos menos relevantes. A seguir apresenta-se o ficheiro contendo a descrição da base de dados de utilizadores:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
<extension vendor-name="FastObjects" key="database" value="BDU"/>
<extension vendor-name="FastObjects" key="schema" value="BDUSchema"/>
    <package name="utilizadores">
        <class name="Utilizador">
            <extension vendor-name="FastObjects"
                  key="index" value="UtilizadorNomeIdx">
                  <extension vendor-name="FastObjects"
                        key="member" value="nome"/>
            </extension>
        </class>
        <class name="Gestor">
            <extension vendor-name="FastObjects"
                  key="index" value="GestorNomeIdx">
                  <extension vendor-name="FastObjects"
                        key="member" value="nome"/>
            </extension>
        </class>
        <class name="Cliente">
            <extension vendor-name="FastObjects"
                  key="index" value="ClienteNomeIdx">
                  <extension vendor-name="FastObjects"
                        key="member" value="nome"/>
            </extension>
        </class>
    </package>
</jdo> .
```

Os métodos disponibilizados pela classe `Utilizador` permitem consultar os valores dos atributos `nome` (`getNome`) e `senha` (`getSenha`), para efeitos de autenticação, e modificá-los (`setNome` e `setSenha`, respectivamente). Os objectos `Cliente` possuem mais um atributo, `activo`, que indica se o utilizador se encontra actualmente ligado ao sistema. Este atributo permite ao gestor do sistema ter acesso à lista de clientes activos em cada instante, percorrendo o *índex* `ClienteNomeIdx` e invocando o método `getActivo` para cada cliente. O valor deste atributo assume o valor `true` quando o

cliente é autenticado, sendo colocado no estado `false` quando o cliente abandona o sistema. Os objectos `Gestor` possuem métodos próprios para alteração e verificação dos dados de utilizador (nome e senha), de forma a poderem, eventualmente, ser inseridos mecanismos distintos de validação, dada a especificidade deste tipo de utilizador.

A aplicação que lida directamente com a base de dados de utilizadores, designada `operBDUtil`, possui cinco métodos públicos, que permitem efectuar as seguintes operações nas bases de dados:

- `ver`: visualização da lista de clientes registados;
- `criar`: criação de cliente;
- `mudar`: modificação da senha de um cliente;
- `eliminar`: eliminação de um cliente;
- `verificar`: verificação do nome e da senha de um cliente.

Os primeiros quatro métodos são usados pelo gestor do sistema, através da aplicação `GestorSistema`, que será analisada mais adiante, enquanto o último é usado pelo Serviço de Autenticação para autenticar os clientes que tentam entrar no sistema.

A seguir apresenta-se um excerto da aplicação `operBDUtil`, onde se podem ver as partes mais relevantes do código utilizado na visualização da lista de utilizadores registados.

```
[…]
// Obtenção do gestor de persistência
PersistenceManagerFactory
pmf=JDOHelper.getPersistenceManagerFactory(pmfProps);
[…]
pm=pmf.getPersistenceManager();
[…]
// Inicio da transacção na base de dados
txn=pm.currentTransaction();
txn.begin();
// Identificação do index a utilizar
Extent ext=pm.getExtent(classe,true);
```

```
Iterator iter=Extents.iterator(ext,”ClienteNomeIdx”);
[…]
// Construção da lista de utilizadores registados
lista=new java.util.LinkedList();
while (iter.hasNext()){
      […]
      lista.add(obj.getNome());
}
txn.rollback();   // Fim da transacção
[…] .
```

A criação de um novo utilizador pressupõe a verificação prévia da sua existência, para evitar conflitos. Note-se, porém, que a base de dados permite a existência de vários utilizadores com o mesmo nome, fazendo-se essa verificação por ser preferível, por questões práticas de utilização do sistema, cada utilizador possuir um nome único. O essencial da codificação do processo de criação apresenta-se a seguir:

```
// Primeiro verifica se o utilizador não está já registado
if(Extents.findKey(iter,nome)==false)
{
      // Utilizador é criado, mas ainda não está activo
      utilizadores.Cliente ucl=new utilizadores.Cliente(nome,senha);
      ucl.setActivo(false);
      pm.makePersistent(ucl);       // Armazena o objecto na BD
      txn.commit();
} .
```

A eliminação envolve a procura do utilizador pelo seu nome – assumindo-se que ele é único, o que é garantido pelo processo de criação – e a invocação do método da API que elimina um objecto persistente, como se pode ver no seguinte excerto da aplicação `operBDUtil`:

```
if(Extents.findKey(iter,nome)){
      pm.deletePersistent(obj);
      txn.commit();} .
```

Na mudança da senha de um utilizador, faz-se também a procura pelo nome e, se esta for bem sucedida, invoca-se o método `setSenha`, passando-lhe como parâmetro a nova senha entretanto introduzida pelo gestor. Note-se que as operações de criação e eliminação de utilizadores e de mudança de senha apenas são permitidas ao gestor através da aplicação de gestão do sistema.

5.5. Base de Dados de Recursos

A base de dados de recursos armazena objectos de duas classes, `Info` e `Aplic`, as quais são subclasses da classe `Recurso`, herdando todos os seus atributos e métodos, acrescentando apenas os elementos que são específicos dos recursos informativos ou das aplicações. Esta hierarquia está representada na Figura 5-6.

Nesta implementação da base de dados de recursos, a classe `Aplic`, que armazena a informação relativa às aplicações, não acrescenta nada relativamente aos atributos e métodos fornecidos pela sua superclasse, `Recurso`, mas optou-se por manter uma estrutura hierárquica, de forma a contemplar possíveis alterações na estrutura da informação a guardar na base de dados. No que diz respeito à classe `Info`, que guarda informação acerca dos recursos informativos disponíveis (ficheiros de informação multimédia), é acrescentado um campo, `autor`, que identifica o utilizador que colocou a informação na base de dados. As alterações nos atributos dos objectos passíveis de ser alterados, `descricao` e `metadados`, é feita através dos métodos `setDescricao` e `setMetadados`.

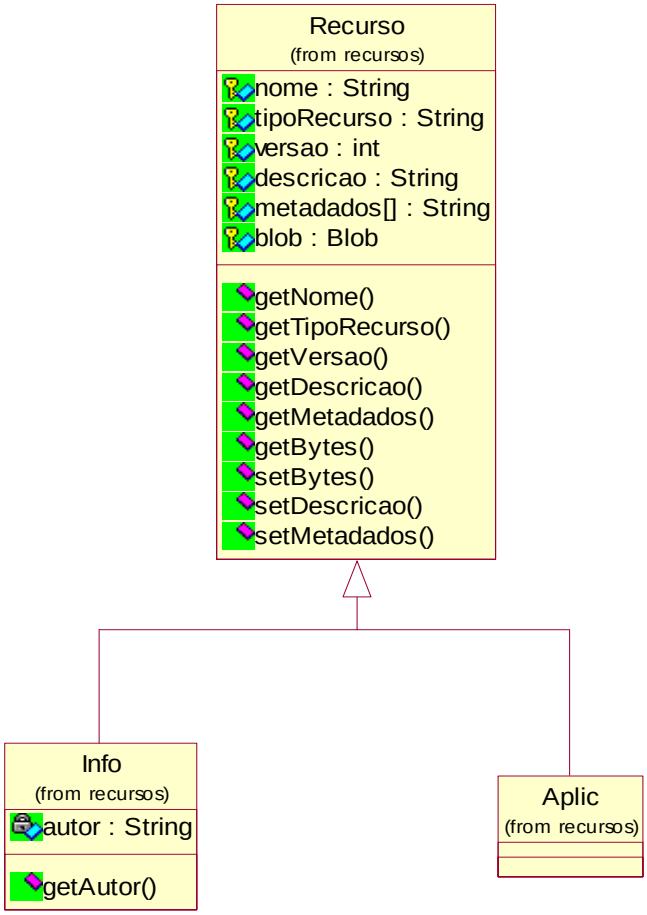

Figura 5-6 – Classes da Base de Dados de Recursos

Os atributos que `Info` e `Aplic` herdam da superclasse `Recurso` são os seguintes:

- `nome`: guarda o nome do recurso – podem existir vários objectos com o mesmo nome, mas de tipos de recursos ou versões diferentes;
- `tipoRecurso`: identifica o tipo de recurso. Por exemplo, se o recurso for uma aplicação, identifica se é Java, C++, etc; se o recurso for um ficheiro contendo informação multimédia, identifica se é MPEG, JPEG, TXT ou outro qualquer tipo de média. Este atributo é do tipo `String`, para não limitar os tipos de recursos a nenhum conjunto restrito, dando liberdade ao autor do recurso para escolher a palavra que melhor o identifique;
- `versao`: indica qual a versão do recurso, que é particularmente importante no contexto das aplicações cooperativas, uma vez que existem

potencialmente vários utilizadores a manipular o mesmo recurso informativo;

- `descricao`: pequeno texto, contendo uma breve descrição do recurso;
- `metadados`: apesar da designação, não armazena informação estruturada segundo algum esquema predefinido, como aconteceria se se usasse RDF [159], Dublin Core [160] ou MPEG-7 [161]. Nesta base de dados apenas é armazenado um conjunto de palavras-chave, para efeito de catalogação do recurso e para poderem ser efectuadas pesquisas com alguma informação acerca desses recursos. A designação de metadados é mantida tendo em vista uma possível evolução futura;
- `blob`: guarda o recurso – aproveita a possibilidade das bases de dados de objectos armazenarem objectos "grandes", habitualmente designados por *blob*'s (*Binary Large OBjects*), para armazenar não só a informação acerca do recurso, como também o ficheiro que o contém.

Os métodos `getNome`, `getTipoRecurso`, `getVersao`, `getDescricao` e `getMetadados` são usados para obter os dados a visualizar, quando se pretende ver a informação relativa a um recurso.

O método `getBytes` permite fazer o carregamento de um recurso que se encontra armazenado na base de dados. Este método acede ao *blob* que corresponde ao ficheiro em questão e carrega o seu conteúdo para uma *stream* binária que entrega à aplicação que invocou o método, para que esta possa reconstruir o ficheiro e manipulá-lo. A codificação deste método apresenta-se a seguir:

```
public byte[] getBytes()
{
      […]
      InputStream in = blob.getInputStream();
      byte[] buffer = new byte[ in.available() ];
      in.read( buffer );
      return buffer;
      […]
} .
```

Da mesma forma, o método `setBytes` permite a gravação na base de dados do ficheiro de um novo recurso, conforme o código a seguir apresentado:

```
public void setBytes(byte[] buffer)
{
      […]
      OutputStream ost = blob.getOutputStream();
      ost.write(buffer);
      ost.flush();
      […]
} .
```

A adição de aplicações à base de dados de recursos só pode ser efectuada pelo gestor do sistema, por intermédio da aplicação de gestão, mas a adição de recursos informativos pode ser feita por qualquer utilizador.

A aplicação que lida directamente com a base de dados de recursos, designada `operBDRec`, permite efectuar operações sobre recursos informativos (usando o *index* `INomeIdx`) e sobre aplicações (usando o *index* `ANomeIdx`). Estas aplicações possuem quatro métodos públicos, que realizam as seguintes operações nas bases de dados:

- `ver`: visualização da lista de recursos disponíveis, na qual é possível seleccionar um recurso, ver os seus detalhes e escolher a opção de o modificar (descrição ou palavras-chave);
- `criar`: criação de um novo recurso;
- `eliminar`: eliminação de um recurso já armazenado;
- `procurar`: pesquisa na base de dados de recursos com um determinado nome ou contendo uma determinada palavra na lista de metadados.

A eliminação de um recurso da base de dados só é permitida ao gestor do sistema, a exemplo do que já foi visto para a base de dados de utilizadores, após selecção da opção de menu correspondente na aplicação de gestão do sistema. O mesmo se passa com a adição de uma aplicação à base de dados, como já foi visto anteriormente, bem

como com a sua modificação. No que diz respeito à procura e à visualização de lista, quer de aplicações, quer de recursos informativos, estão acessíveis a qualquer utilizador.

Os métodos de visualização, criação e eliminação são semelhantes aos apresentados para a aplicação `operBDUtil`. No caso da criação de recursos há ainda que criar o *blob* e as classes de manipulação de ficheiros necessárias ao armazenamento do recurso e invocar o método `setBytes` da classe `Recurso`, conforme se encontra ilustrado no seguinte excerto de código da criação de uma aplicação:

```
public void criar()
{     […]
      Blob blob=new Blob(pm);
      FileInputStream fin=new FileInputStream(ficheiro);
      byte[] buffer=new byte[(int)ficheiro.length];
      int comp=fin.read(buffer);
      […]   // determinação da versão
      obj=new
      recursos.Aplic(nome,tipoRec,versao,descricao,metadados,blob);
      obj.setBytes(buffer);
      pm.makePersistent(obj);
      txn.commit();
} .
```

A versão de um recurso é calculada percorrendo a lista de recursos com o mesmo nome e determinando a versão mais alta dentro dessa lista, sendo a actual igual à determinada mais um.

A seguir apresenta-se um excerto de código para o caso da pesquisa de recursos informativos por palavra-chave, sendo a pesquisa realizada recorrendo a uma linguagem específica de pesquisa em bases de dados de objectos, designada OQL (*Object Query Language*) [162, 163]:

```
public void procurar(String palavra)
{     […]
      String queryString ="define extent ext for recursos.Info; "
            + "select obj from ext as obj where obj.metadados=<ic>$1";
      Query query = pm.newQuery( "org.odmg.OQL", queryString );
      Collection result=(Collection)query.execute(palavra);
      Iterator iter=result.iterator();
      java.util.LinkedList LObjs=new java.util.LinkedList();
      while(iter.hasNext())
      {
            obj =(recursos.Aplic) iter.next();
            LObjs.add(obj);
      }  […]
} .
```

Neste protótipo apenas estão contempladas as pesquisas por nome e por uma palavra-chave, mas podem ser construídas pesquisas segundo outros critérios (tipo de recursos, por exemplo) ou mais complexas, combinando os diversos atributos (por exemplo, pesquisar um recurso com um dado nome e versão superior a determinado valor). Tal, contudo, não foi implementado, por não se mostrar necessário à validação do modelo proposto nesta tese.

5.6. Gestão do Sistema

Como já foi referido antes, muitas das operações que se podem realizar sobre a informação guardada nas bases de dados apenas são permitidas ao gestor do sistema, nomeadamente as operações de criação e destruição de objectos da base de dados. Para efectuar essas operações, o gestor dispõe de uma aplicação específica, designada `GestorSistema`, sendo apresentada na Figura 5-7 a sua janela de entrada. Esta aplicação possui vários menus onde é possível seleccionar as diversas operações de gestão do sistema, sejam elas operações de gestão de utilizadores – lista, criação, eliminação e mudança de senha -, ou operações de gestão de recursos – lista, criação e eliminação de aplicações ou recursos informativos. Possui ainda, no menu “Gestor”, uma opção de mudança da senha do gestor. Dada a importância das operações

disponíveis na aplicação de gestão do sistema, somente o gestor a pode usar, sendo exigida a sua autenticação quando se tenta executá-la.

A aplicação de gestão do sistema utiliza directamente as aplicações `operBDUtil`, e `operBDRec` para executar as operações disponíveis sobre a informação guardada nas bases de dados.

A visualização da lista de utilizadores registados é feita mostrando todos os objectos existentes no *index* `ClienteNomeIdx` e pode ser vista na Figura 5-8. Uma janela semelhante é apresentada quando se escolhe qualquer uma das opções de visualização de lista recursos. A visualização da lista de utilizadores activos faz-se procurando nesse mesmo *índex* os utilizadores cujo campo activo assume o valor `true`, sendo apresentada uma janela semelhante à da Figura 5-8.

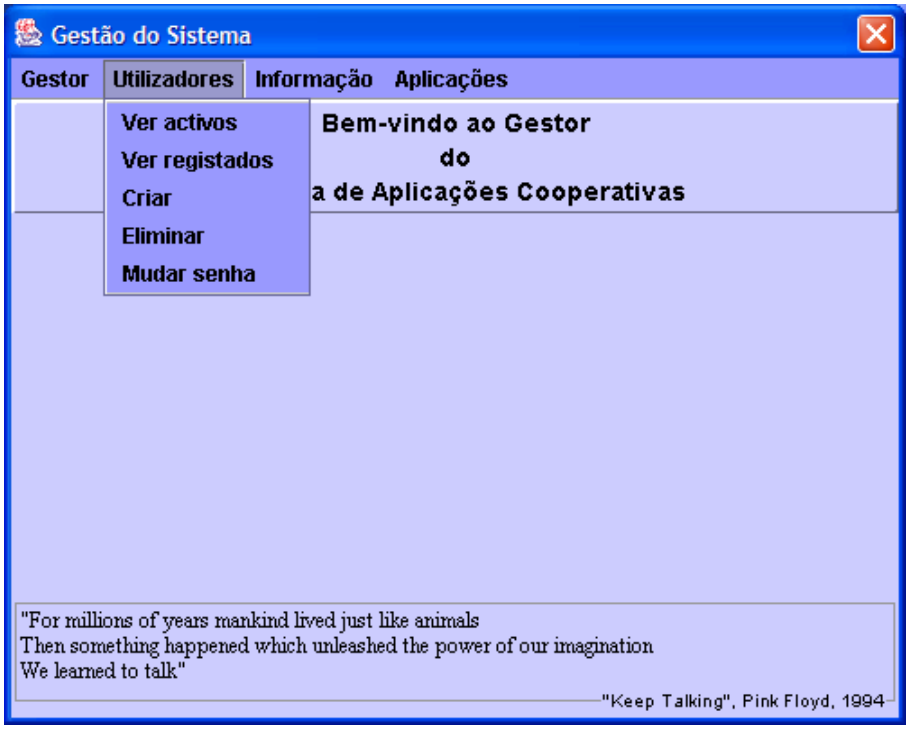

Figura 5-7 – Aplicação de gestão do sistema

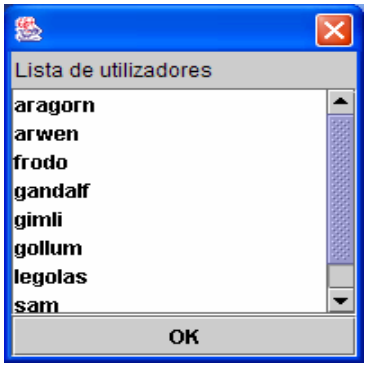

Figura 5-8 – Lista de utilizadores registados

A janela apresentada quando se escolhe a opção de criação de utilizador é semelhante à janela de autenticação (Figura 5-3), residindo a única diferença no texto que é apresentado antes dos campos de inserção de dados.

Para a eliminação de um utilizador ou para a alteração da sua senha é apresentada a janela da Figura 5-9.

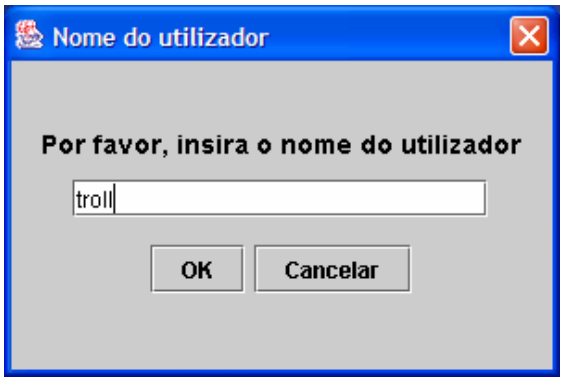

Figura 5-9 – Janela de eliminação de utilizador e de mudança de senha

A Figura 5-10 mostra a janela que é apresentada quando se pretende inserir um novo recurso, seja ele informativo ou uma aplicação.

Figura 5-10 – Janela de adição de recurso

O botão "Escolher" permite seleccionar o ficheiro que contém o recurso que se pretende adicionar, sendo o nome automaticamente inserido de acordo com o nome do ficheiro. Esta opção condiciona o nome do objecto ao nome do ficheiro, mas permite homogeneizar os nomes de diferentes versões do mesmo recurso. Os botões "Adicionar", "Remover" e "Listar" permitem manipular as palavras-chave referentes ao recurso em questão. Assim, o botão "Adicionar" permite acrescentar uma nova palavra-chave à lista já existente, o botão "Remover" permite consultar a lista de palavras-chave e retirar uma e o botão "Listar" permite visualizar a lista actual de palavras-chave associadas ao recurso. O botão "Limpar" permite limpar todos os dados inseridos nos campos até ao momento e o botão "Guardar" torna a informação persistente, efectivando o seu armazenamento na base de dados, incluindo o ficheiro que contém o recurso.

Para a eliminação de um recurso, é apresentada a lista de recursos existentes, selecciona-se o recurso pretendido e é efectivada a eliminação, pelo processo já descrito aquando da apresentação das aplicações `operBDUtil` e `operBDRec`.

A possibilidade que a API JDO oferece para a captura de eventos ocorridos na base de dados, permite que essa informação possa ser usada para fins de gestão, fazendo a monitorização da actividade das bases de dados. Os eventos ocorridos no Serviço de

Eventos, que veremos mais adiante, também podem ser usados para fins de monitorização do sistema. Porém, estas possibilidades de monitorização não se encontram implementadas, por não serem necessárias à validação do sistema, não se justificando o tempo dispendido no seu desenvolvimento.

5.7. Repositórios

Os serviços de Repositório de Aplicações e de Repositório de Informação propostos no modelo de criação de serviços cooperativos possuem muitas características semelhantes. Essa grande semelhança entre ambos verifica-se igualmente no que diz respeito às funcionalidades dos serviços de repositório que foram implementados, mas optou-se por os manter separados, de forma a contemplar uma evolução futura que conduza à sua divergência funcional. Na Figura 5-11 pode-se ver o diagrama de classes dos serviços de Repositório. As aplicações `listaInfo` e `listaAplics` são as aplicações que lidam directamente com esses serviços, identificados por `repInfo` e `repAplic`, respectivamente.

A aplicação `listaAplics` é usada independentemente de outras aplicações, para proceder à visualização e carregamento das aplicações disponíveis na base de dados. A Figura 5-12 mostra a janela de visualização dessas aplicações, sendo necessário seleccionar a aplicação desejada e pressionar o botão “OK” para proceder ao seu carregamento. Este faz-se recorrendo às classes `java.util.jar.JarFile` e `java.io.File`. Uma aplicação Java que contenha elementos gráficos é habitualmente composta por várias classes, as quais são armazenadas em ficheiros separados. Esta característica está na origem da decisão de juntar todos os ficheiros contendo classes de uma aplicação num único ficheiro **JAR**, de forma a facilitar o seu carregamento a partir do servidor. A classe `JarFile` possui depois os métodos que permitem a extracção das várias classes contidas no ficheiro JAR.

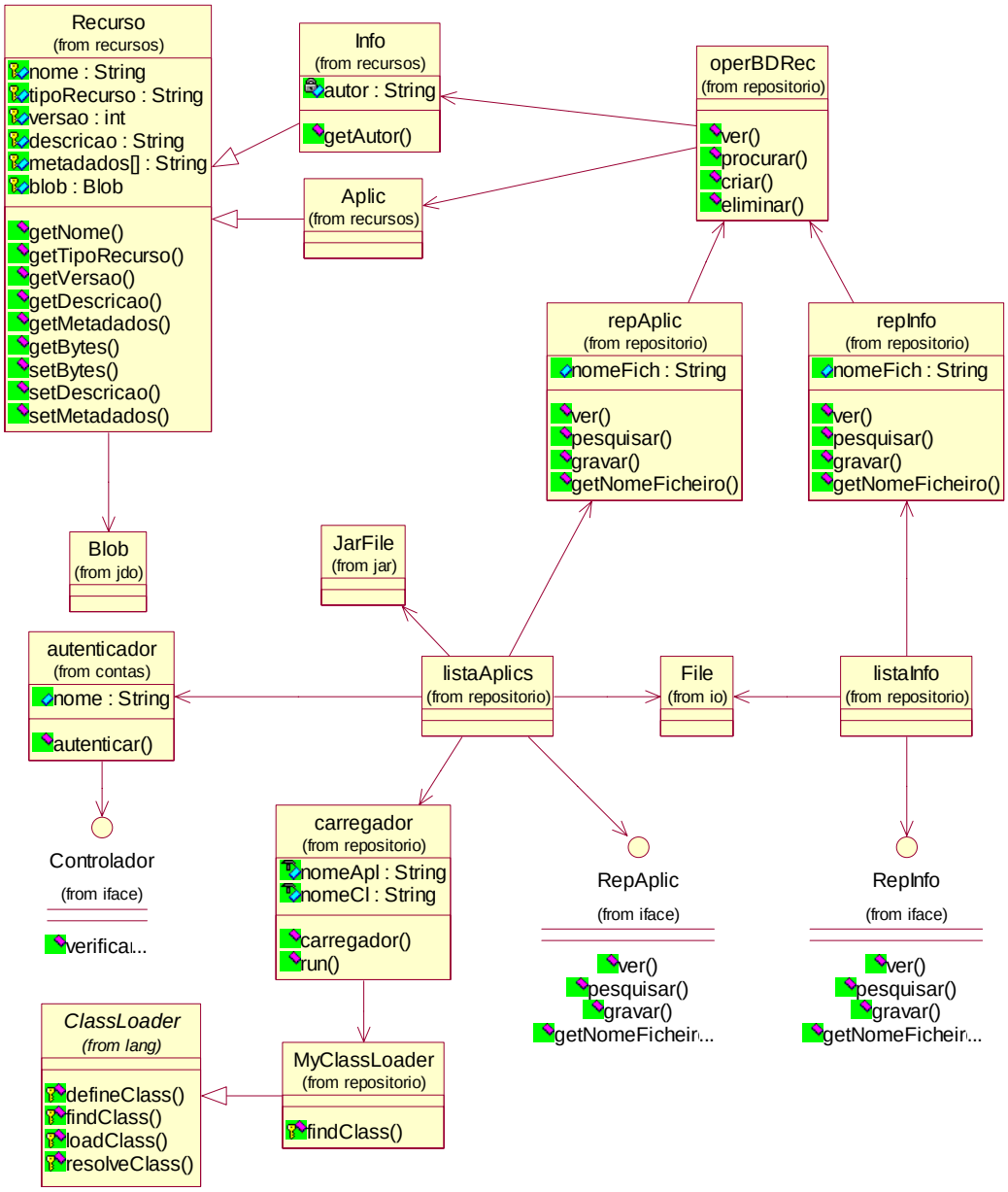

Figura 5-11 – Diagrama de classes dos Serviços de Repositório

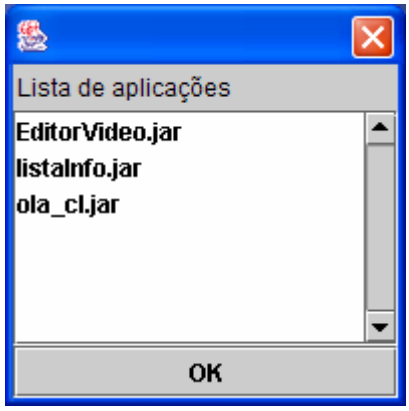

Figura 5-12 – Lista de aplicações disponíveis

Depois de extraídas, as classes são executadas recorrendo a uma importante funcionalidade do Java, o ***Class Loader***, que permite a aplicações Java fazerem o carregamento de classes em *runtime*. Para tal, o programador tem que implementar o seu próprio *Class Loader* (que neste ambiente cooperativo se chama **`MyClassLoader`**), o qual deve ser declarado como uma subclasse da classe `java.lang.ClassLoader`, e implementar o método **`findClass`** que esta última classe define, de forma a fazer a leitura da classe a partir do ficheiro que a contém. A classe `carregador` cria uma instância da classe `MyClassLoader` para cada classe que pretende carregar e invoca nessa instância o método **`loadClass`**, que não foi implementado pelo programador, mas herdado da classe `java.lang.ClassLoader`. Este método usa o método `findClass` para carregar para memória a classe desejada e posteriormente executar o seu método **`main`**, dando início à execução da aplicação.

Os serviços de Repositório de Informação e de Repositório de Aplicações disponibilizam nas suas interfaces os seguintes métodos, os quais usam os serviços da classe `operBDRec` para aceder à base de dados:

- `ver`: permite ver a lista de recursos disponíveis. A partir da visualização, o utilizador pode seleccionar um recurso, ver os seus detalhes e proceder ao seu carregamento. Este método devolve uma *stream* binária (`byte[]`) contendo o recurso ou `null`, caso o utilizador não tenha seleccionado nenhum;
- `pesquisar`: permite procurar recursos que possuam uma dada palavra na sua lista de palavras-chave;

- `gravar`: permite que as aplicações cooperativas gravem na base de dados novos recursos, produzidos durante a sua actividade;
- `getNomeFicheiro`: devolve o nome do ficheiro seleccionado, visto que o método `ver` apenas devolve o conteúdo binário do ficheiro, sendo o seu nome necessário no processo de gravação local.

A aplicação `listaInfo` é usada pelas aplicações cooperativas para aceder à informação remota armazenada na Base de Dados de Recursos. Para poder carregar os ficheiros contendo a informação, utiliza a classe `java.io.File`, que se encontra também representada no diagrama. A Figura 5-13 mostra a janela que é apresentada no caso da visualização da lista de recursos informativos que se encontram disponíveis na base de dados.

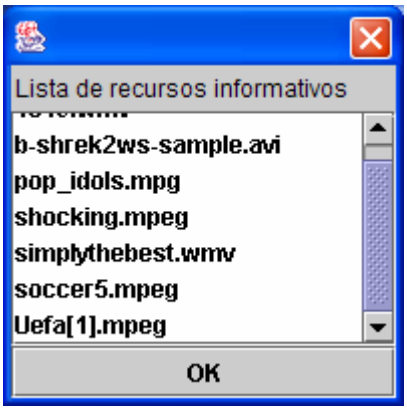

Figura 5-13 – Lista de recursos informativos

Após se ter seleccionado um recurso na lista e se ter carregado no botão "OK", é apresentada uma janela contendo os detalhes desse recurso, como pode ser visto na Figura 5-14 para o caso de um recurso informativo. Para uma aplicação, a única diferença é o facto de o campo de autor não se encontrar preenchido. A opção de apresentar uma lista contendo apenas os nomes dos recursos, sem mostrar mais nenhuma informação, foi tomada por uma questão de simplicidade e por permitir um carregamento mais rápido dessa lista, sendo sempre possível ver os detalhes dos recursos após a sua selecção ou efectuar pesquisas por palavra-chave.

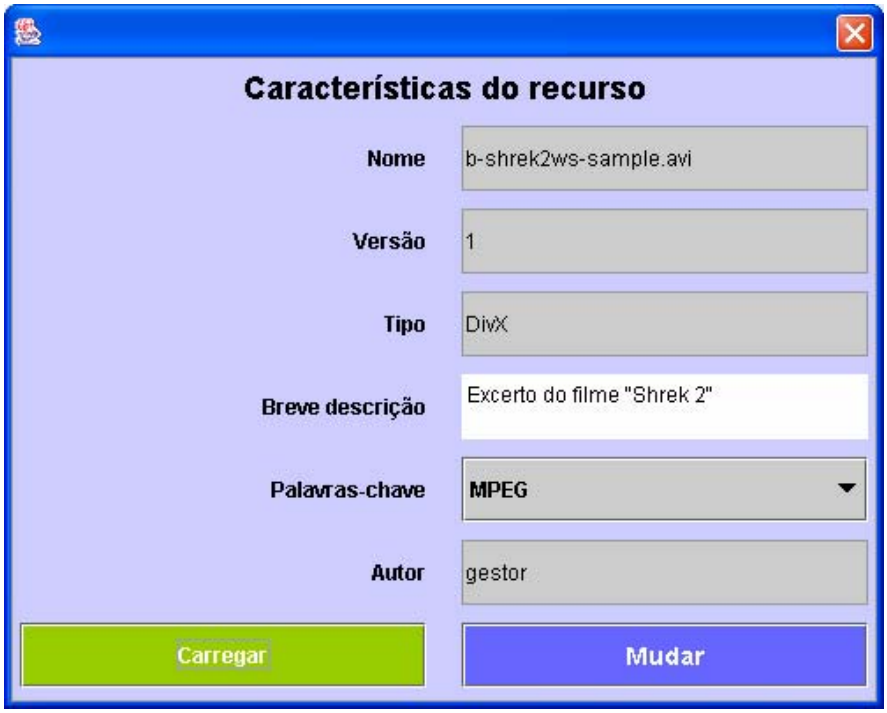

Figura 5-14 – Janela de detalhes de um recurso

5.8. Notificação de Eventos

Existem já APIs específicas que permitem a programação de sistemas de notificação de eventos, como é o caso do JSDT (Java *Shared Data Toolkit*). No entanto, uma solução usando apenas uma dessas APIs fica presa a uma determinada linguagem de programação, não sendo possível que aplicações escritas em linguagens de programação diferentes possam usufruir do sistema, limitando a sua utilização. Por exemplo, usando-se apenas o JSDT não é possível propagar os eventos de uma aplicação Java a uma aplicação C++ e vice-versa. Outro exemplo da existência de sistemas de notificação de eventos é o da arquitectura CORBA, na qual se encontra normalizado um serviço de eventos (*CORBA Event Service*), mas que nem todas as implementações da arquitectura suportam.

Na notificação de um evento, segundo o modelo *publish-subscribe*, é enviada uma mensagem simples para cada utilizador que registou o interesse em ser notificado aquando da ocorrência de eventos desse tipo. Os *web services* adequam-se perfeitamente à troca de mensagens simples, oferecendo ainda a interoperabilidade

necessária à troca de mensagens entre aplicações de plataformas diversas. Contudo, não se encontra normalizado nenhum serviço de notificação de eventos, nem existem ainda no mercado implementações suficientemente robustas e/ou documentadas. No entanto, pode-se adoptar um modelo misto, em que os *web services* fornecem a desejada interoperabilidade e simplicidade na troca de mensagens, existindo por trás um sistema de eventos baseado numa API de uma linguagem de programação, que fornece os mecanismos que, por si só, os *web services* não conseguem disponibilizar. Neste ambiente cooperativo usa-se precisamente a combinação de um Serviço de Eventos com um sistema de distribuição de eventos baseado no JSDT.

O JSDT permite a uma aplicação, que deseje reflectir as suas acções noutras aplicações, criar uma sessão de eventos e registá-la no *Registry*, que guarda informação acerca das sessões de eventos existentes e respectivos clientes. Uma aplicação que deseje receber um determinado tipo de notificações de eventos de outras aplicações regista-se na sessão respeitante a esses eventos e cria o respectivo canal para a troca de informação, após também se ter registado como cliente de eventos. Para ser cliente de eventos, uma aplicação tem, antes de mais, que declarar que implementa a interface `Client` definida pelo JSDT e que disponibiliza os métodos `getName` e `authenticate`. Para além disso, tem que se declarar como consumidor de eventos, implementando a interface `ChannelConsumer` e o seu método `dataReceived`, o qual faz a recepção assíncrona dos eventos. A informação que se deseja propagar é encapsulada na classe `Data`, que possui construtores para dados do tipo `byte`, `String` e `Object`, e enviada para todos os clientes da sessão, para os outros clientes ou para um cliente específico – métodos `sendToAll`, `sendToOthers` e `sendToClient`, respectivamente. Uma aplicação pode-se registar em várias sessões, para receber vários tipos de eventos, e pode também criar várias sessões, para propagar diferentes tipos de eventos.

No ambiente cooperativo aqui apresentado, as aplicações produzem eventos que reflectem as suas actividades cooperativas e estão também interessadas em receber as notificações dos eventos produzidos por outras aplicações. No entanto, não são as aplicações que se registam directamente como produtoras e consumidoras de eventos, isso é feito pelo Serviço de Notificação de Eventos, que irá intermediar toda o processo, por uma questão de interoperabilidade. Desta forma, cada aplicação cria uma instância do Serviço de Notificação de Eventos, a qual se encarregará de lhe entregar as

notificações que lhe dizem respeito. Ou seja, o Serviço de Notificação de Eventos comporta-se como produtor e consumidor de eventos e o Sistema de Eventos fornece os meios para a distribuição dos eventos pelas várias instâncias do serviço. Contudo, os eventos reflectem acções produzidas pelas aplicações e destinam-se igualmente a estas, sendo necessário estabelecer uma correspondência entre as acções e os eventos. Esta correspondência é feita através da troca de mensagens assíncronas ente a aplicação e o Serviço de Notificação de Eventos. A Figura 5-15 mostra o diagrama de classes do Serviço de Notificação de Eventos.

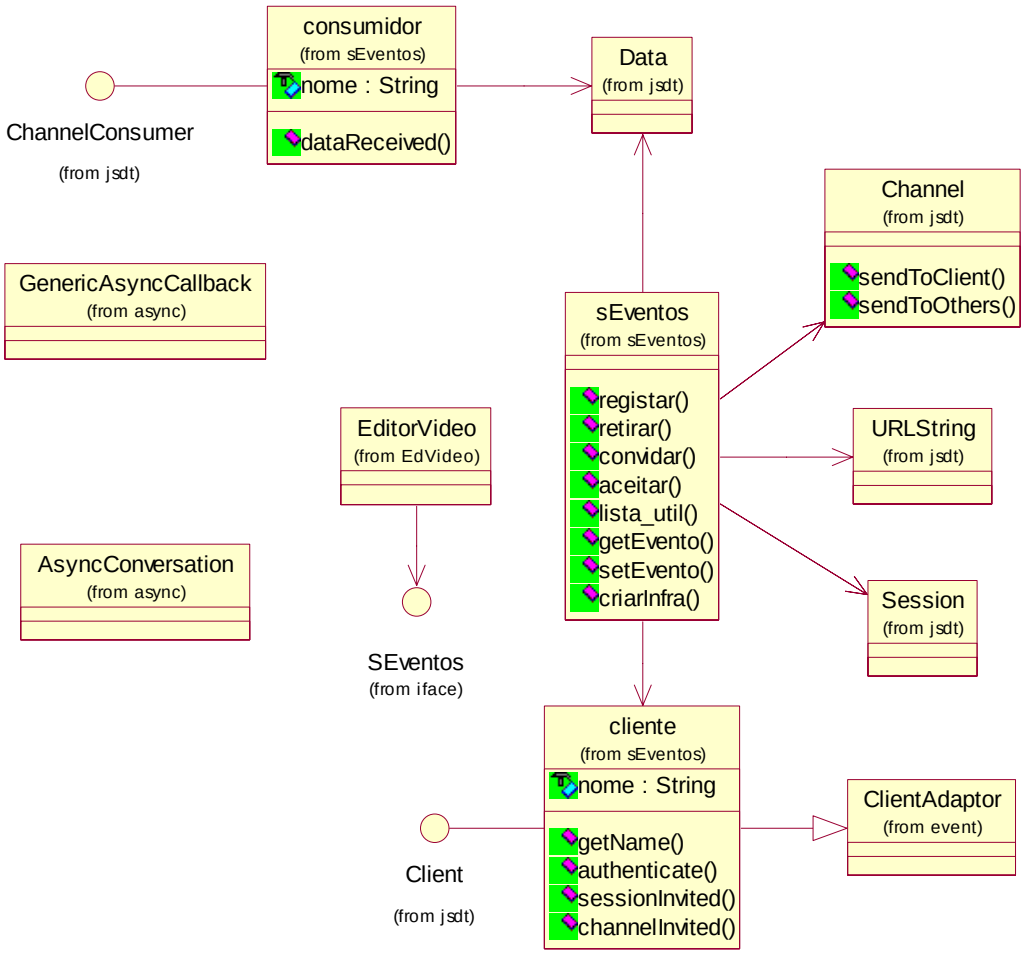

Figura 5-15 – Diagrama de classes do Serviço de Eventos

O JSDT permite escolher um de três protocolos para a propagação dos eventos: *sockets* TCP/IP, HTTP ou LRMP (*Lightweight Reliable Multicast Package*). A escolha do protocolo HTTP tem a vantagem de facilitar a utilização do JSDT em ambientes com

firewalls, tirando partido da técnica *http tunneling*. O sistema de eventos que aqui se apresenta tem todos os componentes localizados no mesmo computador. Contudo, se se pretender uma solução com um maior grau de distribuição, a utilização do protocolo http é adequada, tendo em vista a eventual necessidade de tirar partido do *http tunneling* para atravessar *firewalls*.

O WASP permite criar automaticamente métodos assíncronos aquando da geração da *proxy* do serviço. Assim, para um método `xpto` são criados automaticamente os métodos `benginXpto` e `endXpto`, que permitem, respectivamente, fazer uma invocação assíncrona e recolher mais tarde os resultados. Para tirar partido desta característica, a aplicação deve incluir uma classe que estende a classe WASP `GenericAsyncCallback` e incluir nessa classe o método `onResponse`, que é invocado automaticamente sempre que se encontram disponíveis resultados de uma invocação assíncrona.

Um utilizador do sistema pode estar ou não interessado em entrar numa sessão de trabalho cooperativo, pelo que as aplicações devem ser configuráveis para trabalhar isoladamente ou em ambiente cooperativo. Quando a aplicação é iniciada não entra logo em modo cooperativo, mas regista-se como cliente de eventos de sistema, para que possa ser convidada a participar numa sessão cooperativa ou convidar outro utilizador. Para poder convidar outro utilizador para uma sessão cooperativa, o utilizador consulta a lista de utilizadores que estão activos, ou seja, que estão registados na sessão de sistema. Ao seleccionar um utilizador, é-lhe enviada uma mensagem de convite através do canal de eventos de sistema. O utilizador convidado pode aceitar ou rejeitar esse convite, sendo enviada de volta a respectiva mensagem através do mesmo canal. Se o convite for aceite, ambos os utilizadores se registam no canal de eventos específico da aplicação, caso ainda não o tenham feito (podem já estar envolvidos noutra sessão cooperativa do mesmo tipo). Uma vez em modo cooperativo, a aplicação recebe também todas as notificações de eventos respeitantes ao canal de eventos específicos da aplicação. Cada utilizador pode envolver-se numa sessão cooperativa com vários utilizadores, sendo os eventos propagados a todos os utilizadores que estiverem registados na sessão cooperativa. Para além disso, podem ser estabelecidas várias sessões com utilizadores distintos, ou seja, um utilizador A pode ter uma sessão com um utilizador B e uma outra sessão com os utilizadores C e D, não se misturando os eventos das diferentes sessões.

A interface do protótipo do Serviço de Notificação de Eventos apresenta os seguintes métodos:

- `registar`: permite a uma aplicação registar o interesse na recepção de eventos de sistema;
- `retirar`: é invocado por uma aplicação que não quer receber mais eventos específicos da aplicação;
- `lista_util`: devolve a lista de utilizadores que se encontram activos no momento, de forma a poderem ser convidados para uma sessão cooperativa;
- `convidar(String nome_c, nome_s)`: permite convidar um utilizador (`nome_c`), seleccionado na lista de utilizadores activos, para uma sessão cooperativa (sessão identificada por `nome_s`);
- `aceitar(boolean aceite,[...])`: é invocado por um utilizador que foi convidado, indicando se aceitou (`aceite=true`) ou se rejeitou (`aceite=false`) a proposta;
- `getEvento`: permite receber eventos de um dado canal. A aplicação utiliza assincronamente este método, invocando o método `beginGetEvento` após registar-se num canal e sempre que recebe uma notificação, para que possa receber a resposta assíncrona seguinte. A recepção da resposta é feita invocando o método `endGetEvento`;
- `setEvento(String ev, String nome_cl)`: é invocado para comunicar a ocorrência de um evento identificado por `ev`, de forma a ser propagado às outras aplicações interessadas ou à identificada por `nome_cl`.

A seguir apresenta-se um excerto da codificação da aplicação de edição cooperativa de vídeo que implementa a recepção assíncrona de eventos:

```
static class MyCallbackClass extends GenericAsyncCallback
{
      SEventos serv;

      public MyCallbackClass(SEventos serv)
      {
            super();
            this.serv=serv;
      }

      public void onResponse
      (org.systinet.wasp.async.AsyncConversation asyncConversation)
      {
            String evento=serv.endGetEvento(asyncConversation);
            […]
            // Tratamento da informação recebida
            […]
            AsyncConversation async1=serv.beginGetEvento(true);
            MyCallbackClass callback1=new MyCallbackClass(serv);
            async1.setCallback(callback1);
            AsyncConversation async2=serv.beginGetEvento(false);
            MyCallbackClass callback2=new MyCallbackClass(serv);
            async2.setCallback(callback2);
        }
    } .
```

Na codificação apresentada são feitas duas invocações assíncronas no final, para que a aplicação possa continuar a receber eventos, quer do canal de eventos de sistema, quer do canal de eventos específicos da aplicação.

O tipo de evento, referido no registo de interesse em eventos por parte de uma aplicação, é passado como parâmetro da invocação, sob a forma de um conjunto de caracteres (*String*). Da mesma forma, o valor do evento também é passado como um parâmetro do tipo `String`, na invocação da operação de notificação. Assim, consegue-se que exista uma única interface para todo o tipo de aplicações cooperativas, o que

confere versatilidade ao serviço. A decisão de propagar ou não um determinado evento para uma dada aplicação é tomada no lado do sistema de eventos, continuando as aplicações a receber apenas as notificações nas quais estão interessadas. Por sua vez, as aplicações processam os eventos que recebem de diferentes maneiras, de acordo com o tipo de evento em questão.

Quando uma aplicação produz um evento que deseja propagar, invoca o método `setEvento` na sua instância do Serviço de Notificação de Eventos, que se encarrega de o comunicar ao Sistema de Eventos, o qual, por sua vez, o distribui pelas outras instâncias do serviço, para que estas os possam entregar às outras aplicações.

5.9. Aplicação de edição cooperativa de vídeo

Para testar a implementação do modelo de criação de serviços cooperativos que se apresenta neste capítulo, construiu-se um protótipo de uma aplicação que permite a utilizadores, situados em locais distintos, cooperarem em processos de edição de vídeo, partilhando algumas actividades de edição e visualização de informação audiovisual.

Os sistemas de edição de vídeo mais antigos envolviam a localização de fitas de vídeo no arquivo e o seu transporte para a sala de edição. O trabalho de edição era efectuado recorrendo a equipamentos analógicos e no fim era necessário devolver as fitas ao arquivo, sendo o resultado da edição gravado numa nova fita.

Com a evolução que a indústria de computadores sofreu nas últimas décadas, tornou-se possível a utilização de técnicas de manipulação digital de vídeo [164]. Numa primeira aproximação, as sequências contidas nas fitas são convertidas para formato digital e armazenadas e manipuladas em estações de trabalho especializadas – estações de edição não-linear de vídeo [165]. De qualquer forma, o original e o resultado da edição são mantidos em formato analógico, apenas há uma manipulação digital do vídeo. Mais recentemente, os desenvolvimentos dos equipamentos de aquisição de vídeo e áudio, das redes de computadores e das técnicas de compressão de informação, permitiram evoluir para o conceito de estúdio digital, onde se utiliza tecnologia digital em toda a cadeia de produção de vídeo, desde a aquisição até à transmissão, passando pelo armazenamento e processamento [166-170].

Uma componente importante num estúdio de produção de vídeo é a aplicação de edição, a qual permite a produção de um novo programa a partir da composição de sequências de vídeo pré-gravadas. Nas aplicações actuais de edição não-linear de vídeo digital, apenas um único utilizador pode usar o sistema em cada instante. Qualquer necessidade de colaboração por parte de outros profissionais exige a sua presença física, ou então a realização de uma chamada telefónica, por exemplo. Na produção de reportagens remotas, os jornalistas têm que transportar uma mala de edição e transmitir para o estúdio o resultado da edição. A disponibilidade de uma ferramenta que possibilite que vários utilizadores partilhem simultaneamente o mesmo ambiente de edição, estando localizados no estúdio ou noutro local qualquer e usando um simples computador pessoal com acesso à Internet, pode produzir resultados interessantes. Parece assim bastante atractiva, por razões operacionais e por razões financeiras, a possibilidade de dispor de uma aplicação de edição cooperativa de vídeo digital.

A edição de vídeo envolve geralmente a manipulação de diversos clipes, dos quais se destacam pedaços para eliminar, alterar a ordem ou combinar com pedaços de outros clipes. Podem ainda ser acrescentadas pistas de áudio, transições, legendas ou efeitos especiais. O protótipo da aplicação de edição cooperativa de vídeo que aqui se apresenta é bastante simples no que diz respeito às funcionalidades oferecidas, pois o objectivo não era construir uma aplicação profissional mas apenas testar a validade das soluções propostas no modelo de criação de serviços cooperativos, dado que o tempo e os recursos necessários para uma implementação completa não são compatíveis com os propósitos de uma Tese de Doutoramento. Assim, as funcionalidades de edição implementadas reduzem-se à definição de pontos de corte num clipe, enquanto se visualiza o clipe original: o ponto IN define o ponto no clipe original a partir do qual se quer definir um novo clipe; o ponto OUT indica o ponto no clipe original que corresponde ao fim do novo clipe. Para além das operações de edição, o editor cooperativo de vídeo oferece a possibilidade de visualizar o clipe original e de pré-visualizar o resultado da edição.

A escolha de clipes, os pontos de corte e as operações de visualização produzem eventos que são transmitidos aos utilizadores cooperantes. A aplicação de edição cooperativa de vídeo, após a recepção de um valor de um ponto de corte pode atribuir esse valor ao seu equivalente local. No que diz respeito às operações de visualização e à

escolha de clipes apenas são apresentados avisos com a informação da acção tomada pelo parceiro cooperante. Mais uma vez, a escolha dos eventos a serem notificados e as acções a tomar mediante a sua recepção, podem não se revelar as mais adequadas, dada a ausência de um estudo acerca dos eventuais interesses cooperativos dos habituais utilizadores de aplicações de edição de vídeo.

A Figura 5-16 representa a arquitectura interna do protótipo de aplicação de edição cooperativa de vídeo. O módulo de controlo de média fornece as funcionalidades necessárias à visualização das sequências de vídeo originais e editadas, recorrendo para o efeito às operações disponibilizadas pela API JMF. O controlo de acesso ao sistema faz-se através do Serviço de Autenticação, para o qual o editor possui a *proxy* respectiva. O módulo de controlo de recursos informativos possibilita a consulta dos ficheiros multimédia remotos disponíveis na base de dados de recursos, por intermédio do Serviço de Repositório de Informação. Finalmente, o controlo de cooperação faz-se através do Serviço de Notificação de Eventos, fornecendo os meios para que se possam convidar (ou ser convidado por) outros utilizadores para sessões de actividade cooperativa e para partilhar os respectivos eventos.

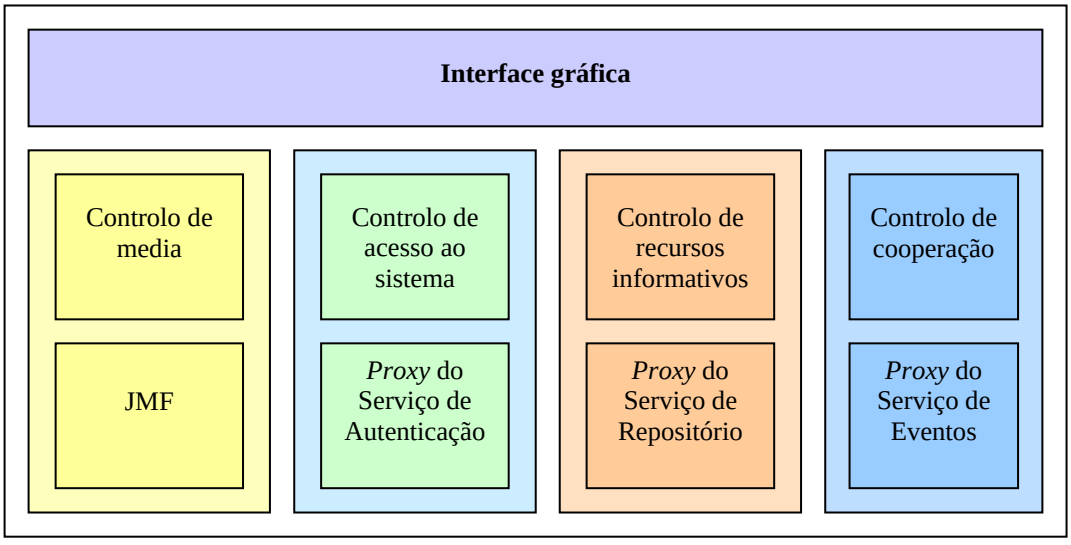

Figura 5-16 – Arquitectura interna da aplicação de edição de vídeo

A Figura 5-17 apresenta o diagrama de classes do protótipo de editor cooperativo de vídeo. Neste diagrama pode-se ver a classe `EditorVideo`, que implementa a

aplicação de edição de vídeo, a qual usa várias classes do JMF (representadas pela cor salmão). A classe `Player` é utilizada para visualizar os ficheiros de vídeo que a aplicação usa, os quais são manipulados com a ajuda da classe `MediaLocator`. A classe `Time` fornece os meios para poder controlar os pontos de início e fim de visualização dos ficheiros de vídeo, característica que é usada para a marcação dos pontos de edição. As classes representadas a amarelo são classes da infra-estrutura de *web services*, que permitem a invocação assíncrona de métodos, utilizada para a manifestação de interesse na recepção de eventos. A cinzento estão representadas as principais classes que servem de base à construção da interface gráfica. Por fim, a azul claro (ciano) estão representadas as classes que implementam os serviços e a verde as aplicações que os usam, nas quais se inclui o editor de vídeo.

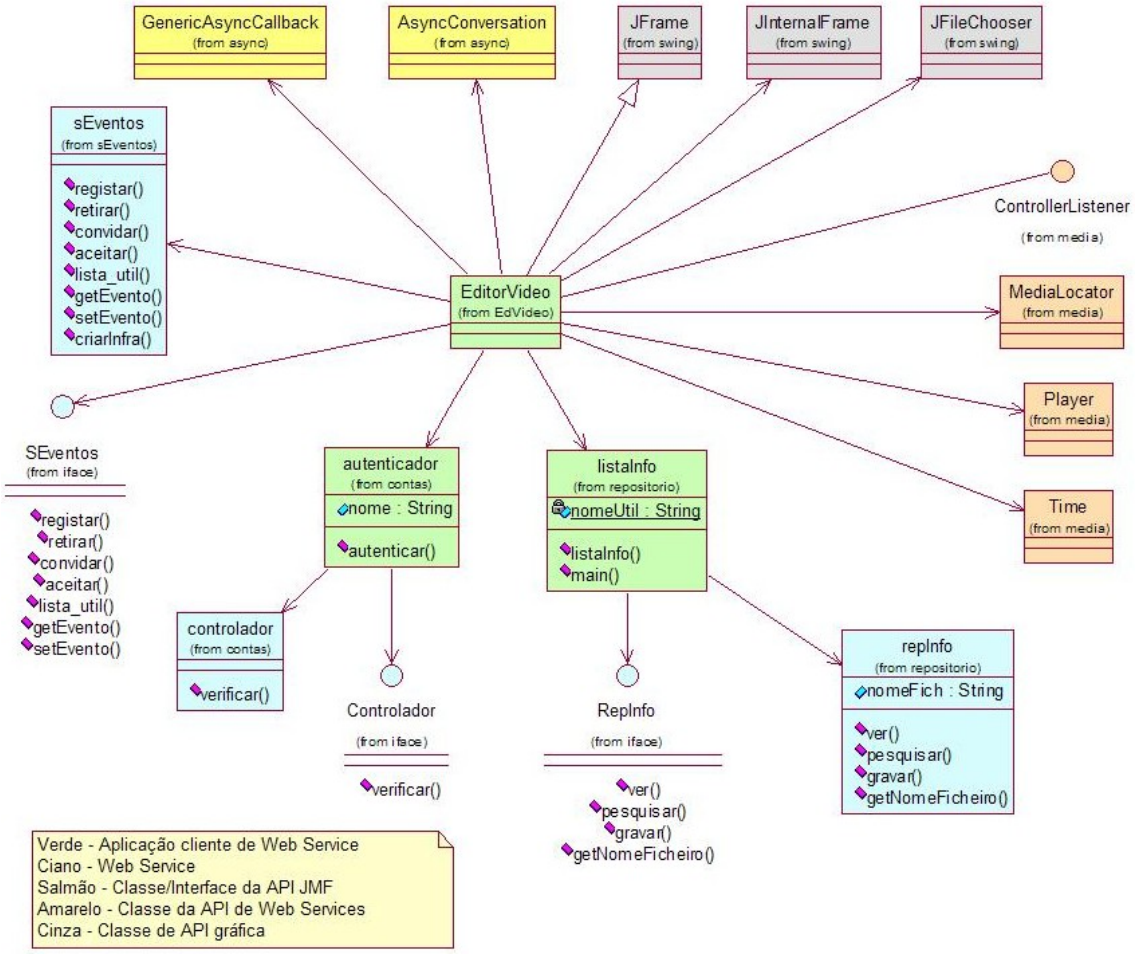

Figura 5-17 – Diagrama de classes do Editor de Vídeo

A Figura 5-18 mostra um diagrama sequencial que representa um exemplo de actividade de um editor de vídeo. Neste exemplo, após a autenticação do utilizador e registo da aplicação no serviço de eventos, é feito o convite a outro utilizador para aderir a uma sessão de actividade cooperativa. Esse utilizador aceita o convite e a recepção dessa aceitação faz com que seja invocado o método `criarInfra` do Serviço de Notificação de Eventos (este método foi omitido anteriormente, por não ser relevante no contexto em que foi apresentado o Serviço de Notificação de Eventos), que estabelece a sessão de eventos de aplicação (caso não exista já) e o canal de eventos respectivo. Posteriormente, o utilizador selecciona a opção "Ver lista de clipes remotos", para escolher um clipe disponível na base de dados. Na visualização da lista de clipes disponíveis, é escolhido um e visualizada a sua informação. A escolha da opção de carregamento do ficheiro faz com que seja invocado o método `getBytes` do recurso informativo, o qual devolve a *stream* binária contendo o ficheiro existente no *blob* armazenado na base de dados. A visualização do clipe e a marcação dos pontos de edição despoletam a emissão de eventos, que são propagados aos utilizadores registados na sessão – neste exemplo, apenas mais um. Quando o utilizador deseja parar a cooperação, selecciona a opção "Parar" do menu "Cooperação", o que provoca a emissão de um evento de sistema, avisador do abandono da sessão por parte do utilizador, evento esse que é propagado aos outros utilizadores por intermédio do Serviço de Notificação de Eventos. A escolha da opção "Guardar no servidor", do menu "Ficheiro", faz com que seja invocado o método `gravar` do Serviço de Repositório de Informação, o qual provoca a invocação do método `criar` da aplicação `operBDRec`, que cria um novo recurso e grava nele o ficheiro resultante da edição, invocando o método `setBytes` no recurso recém-criado. Por fim, o utilizador escolhe a opção de saída da aplicação, o que provoca a invocação do método `retirar` do Serviço de Notificação de Eventos, o qual finaliza os recursos de partilha (canais, sessões e cliente de eventos), sendo finalmente fechada a aplicação.

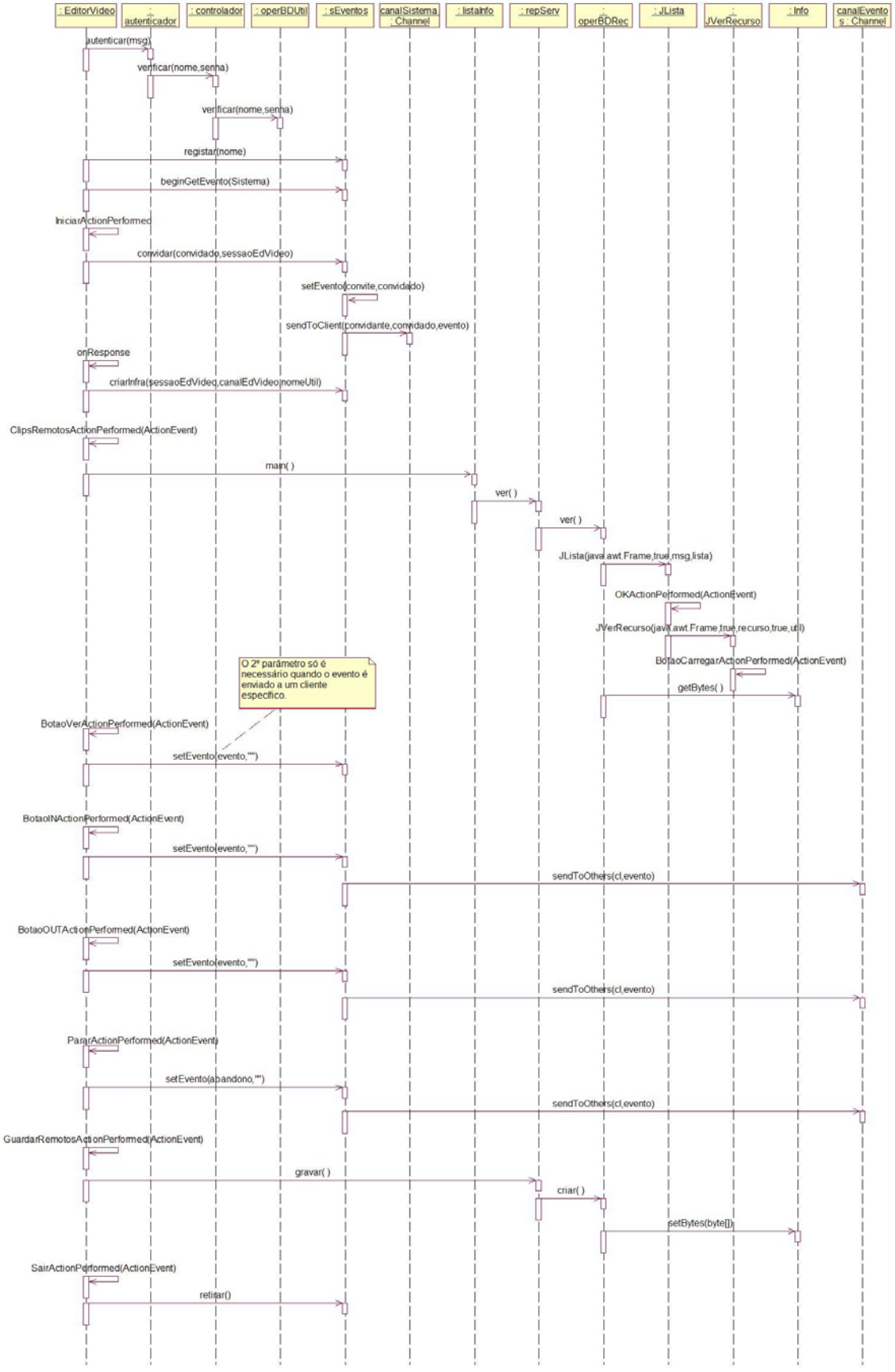

Figura 5-18 – Diagrama sequencial de um exemplo de actividade do editor de vídeo

A Figura 5-19 mostra a janela principal do editor cooperativo de vídeo. Nela se podem ver vários menus, uma janela interna de visualização do clipe e um painel que contém um campo com o nome do clipe actualmente seleccionado e botões para a sua visualização e marcação de pontos de edição. A visualização dos clipes recorre às facilidades disponibilizadas pelo JMF, através da utilização de uma instância da classe `Player`, e fornece funcionalidades básicas de controlo de apresentação do vídeo, permitindo apenas iniciar, suspender ou parar a visualização do clipe e inibir/desinibir o áudio associado. No entanto, o JMF fornece os meios para implementar outras funcionalidades como a visualização em câmara lenta ou em ritmo acelerado.

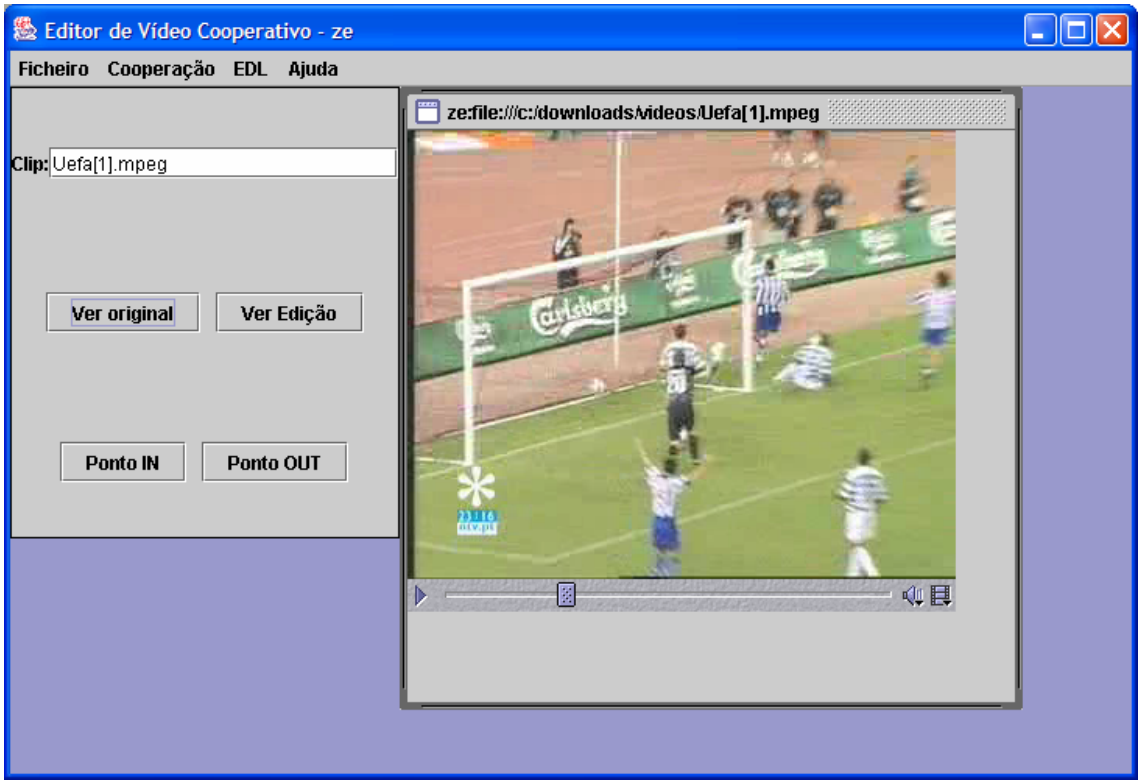

Figura 5-19 – Janela principal do editor de vídeo

Na Figura 5-20 podem-se ver as opções disponíveis nos vários menus da aplicação de edição cooperativa de vídeo. O menu ficheiro possui opções de abertura e gravação de clipes de vídeo, no sistema de armazenamento local ou na base de dados remota, para além da opção de procura de informação por palavra-chave e da opção de saída da aplicação. Os clipes locais são escolhidos numa janela típica de abertura de ficheiro, como se pode ver na Figura 5-21. A escolha da opção de abertura de clipes

remotos provoca a execução da aplicação `listaInfo`, de acesso ao Serviço de Repositório de Informação, no qual invoca o método `ver`. Esta invocação faz com que seja apresentada a janela já vista na Figura 5-13. A execução da aplicação `listaInfo` só ocorre se ela for encontrada no armazenamento local, caso contrário é executada a aplicação `listaAplics`, para proceder ao seu carregamento.

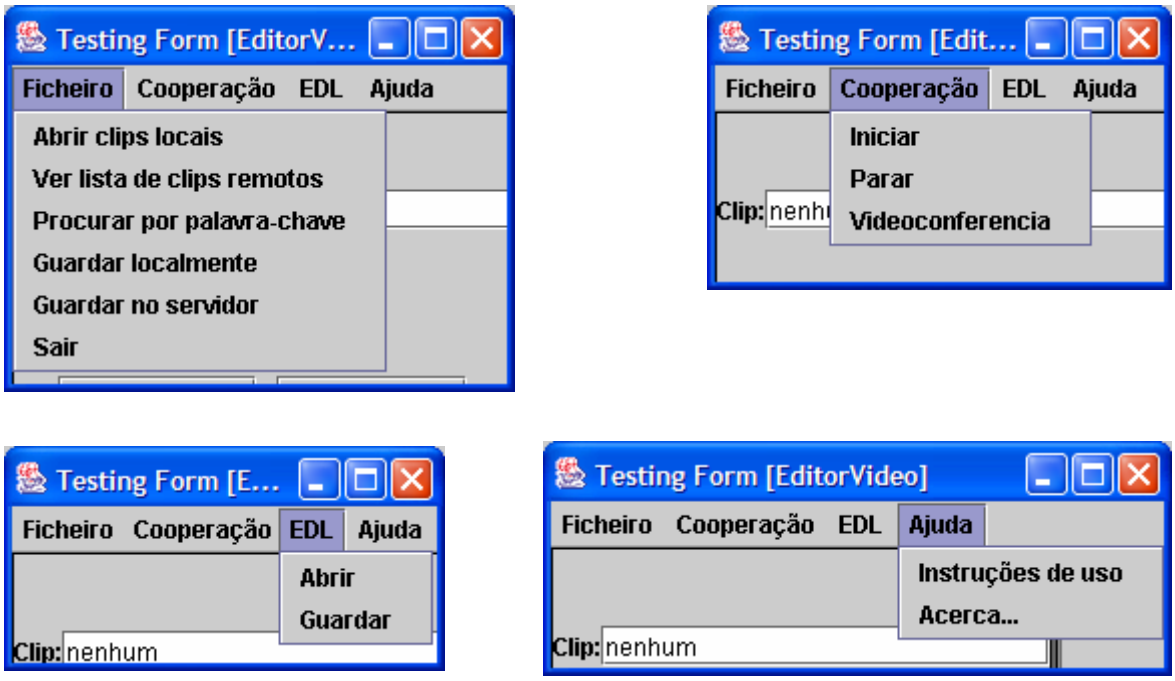

Figura 5-20 – Opções dos menus

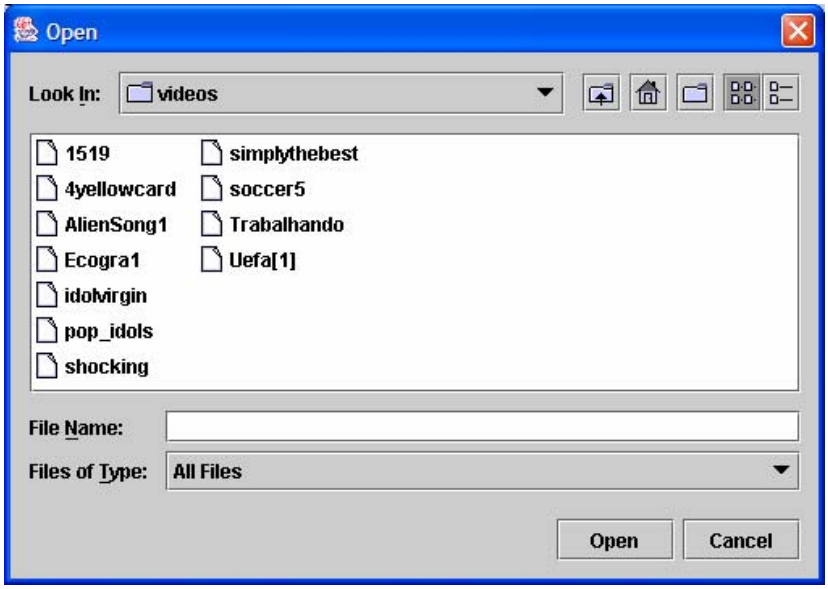

Figura 5-21 – Janela de escolha de clipe

A opção de procura por palavra-chave invoca o método `pesquisar` do Serviço de Repositório de Informação, o qual procura, no campo de metadados dos recursos informativos, a palavra inserida pelo utilizador na janela apresentada na Figura 5-22.

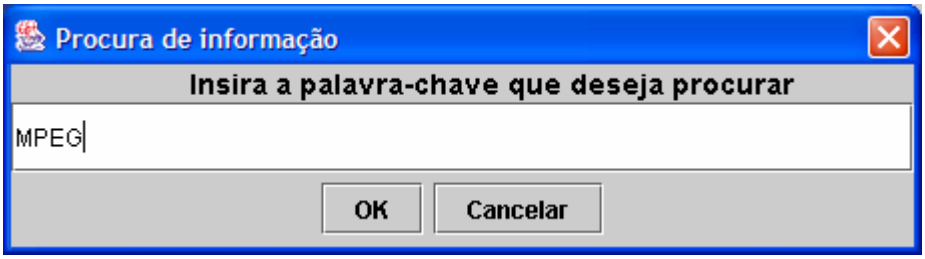

Figura 5-22 – Janela de procura por palavra-chave

As opções "Guardar localmente" e "Guardar no servidor" permitem fazer a gravação dos clipes editados no disco local e na base de dados de recursos, respectivamente. Para tal, é necessário transformar as escolhas dos pontos de edição em cortes efectivos nos clipes de vídeo, o que pode ser efectuado recorrendo a um tipo especial de `Player`, designado `Processor`, que permite efectuar processamento sobre a informação a visualizar. Desta forma, pode-se usar um `Processor` para redireccionar a sequência de vídeo contida entre os pontos de edição para uma *stream* binária, que é passada como parâmetro da invocação do método `gravar` do Serviço de Repositório de Informação, o qual invoca o método `criar` da aplicação `operBDRec`.

O menu de cooperação tem opções para iniciar e parar uma sessão de actividade cooperativa e para iniciar uma sessão de videoconferência com um parceiro cooperativo. Esta última opção não se encontra ainda implementada, devido a restrições temporais, mas, mais uma vez, também não se revela essencial no contexto da validação do modelo de criação de serviços cooperativos. A escolha da opção "Iniciar" provoca a invocação do método `convidar` do Serviço de Notificação de Eventos, o qual apresenta a lista de utilizadores actualmente activos, que pode ser vista na Figura 5-23.

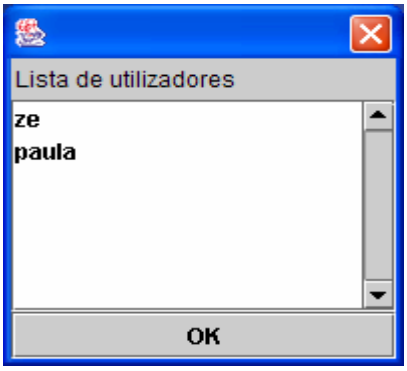

Figura 5-23 – Lista de utilizadores activos

Após seleccionar o utilizador com o qual deseja cooperar, é enviado o evento de sistema "convite" a esse utilizador, o qual é notificado através da janela da Figura 5-24.

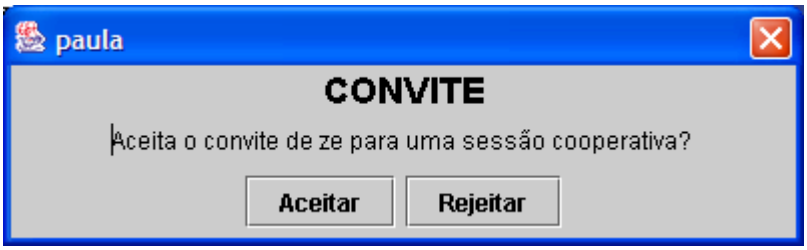

Figura 5-24 – Notificação de convite para cooperação

A aceitação do convite para cooperar produz a notificação apresentada na Figura 5-25 e a sua rejeição produz uma notificação semelhante, mas com um texto referente à rejeição.

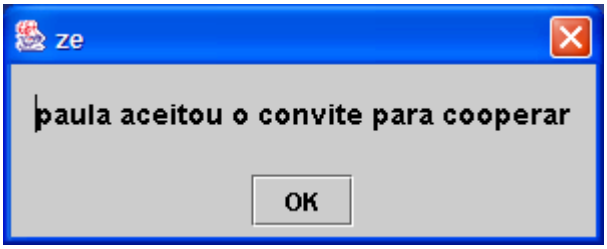

Figura 5-25 – Notificação de aceitação de convite

Quando em modo cooperativo, as acções que o utilizador executa na interface gráfica, que produzam eventos passíveis de serem propagados, fazem invocar o método `setEvento` no Serviço de Notificação de Eventos, o qual se encarrega de os enviar aos

outros utilizadores registados na mesma sessão cooperativa. Os eventos que são propagados entre as instâncias do editor de vídeo são os relativos à definição dos pontos de corte, à visualização do vídeo e à escolha do clipe original. A escolha desses eventos não reflecte, seguramente, o que seria desejável do ponto de vista das exigências profissionais. Tal facto deve-se à escassez de funcionalidades que a aplicação possui e à ausência de um estudo que possibilite a aferição das reais necessidades cooperativas de uma aplicação deste tipo. Esse não é, contudo o âmbito desta Tese de Doutoramento, sendo a escolha das actividades a partilhar irrelevante para a validação do seu modelo de propagação. A Figura 5-26 mostra um exemplo de notificação desse tipo de eventos.

Figura 5-26 – Notificação de evento de actividade cooperativa

Quando um utilizador deseja sair de uma sessão cooperativa, escolhe a opção "Parar" do menu "Cooperação", provocando a invocação do método `terminar` do Serviço de Notificação de Eventos. Este notifica os outros utilizadores registados na sessão cooperativa acerca do abandono do utilizador em questão, através da janela da Figura 5-27.

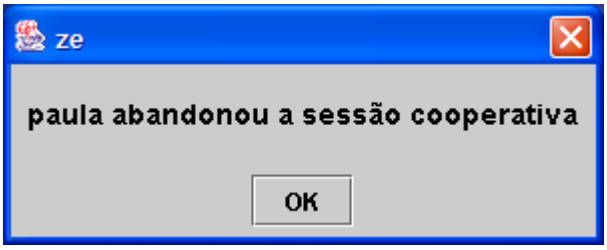

Figura 5-27 – Notificação de abandono da sessão

No menu "EDL" encontram-se opções relativas a operações sobre a lista de decisões de edição (EDL – *Edit Decision List*), a qual costuma ser usada nos sistemas

de edição não-linear de vídeo para descrever textualmente a sequência de operações de edição (pontos de entrada e saída das diversas sequências componentes, transições, efeitos, etc), para mais tarde criar a sequência final. Esta funcionalidade também não se encontra implementada.

5.10. Análise de desempenho do sistema

A validação do modelo proposto nesta tese e do ambiente de suporte à criação de aplicações cooperativas baseado nesse modelo, requer a verificação do correcto funcionamento dos serviços e das aplicações implementadas. Esta verificação pode ser feita observando os aspectos qualitativos do desempenho do sistema, tais como a ocorrência de erros ou de bloqueios durante a execução dos serviços e das aplicações. Se possível, esta análise qualitativa deve ser acompanhada de medições de parâmetros que permitam complementar algumas observações com dados mensuráveis.

A observação do funcionamento global do sistema que foi implementado revelou-se satisfatória. Do ponto de vista dos erros e bloqueios ocorridos no decorrer da execução dos serviços e das aplicações, apenas há a referir o desempenho menos robusto dos componentes audiovisuais, resultante da utilização do JMF. De facto, o uso desta API introduz um acréscimo significativo na carga computacional e requer cuidados especiais com a máquina de estados que controla a visualização da informação audiovisual. No que diz respeito aos atrasos introduzidos pela execução dos serviços de suporte à actividade cooperativa, a observação do funcionamento da aplicação de edição de vídeo com vários utilizadores cooperantes não revelou problemas em regime permanente, ocorrendo apenas algum atraso inicial no estabelecimento das sessões cooperativas, inerente ao próprio processo de convite e aceitação de utilizadores.

Para complementar a análise qualitativa do desempenho do sistema, efectuaram-se alguns testes que permitiram registar informação relativa aos atrasos na propagação de eventos cooperativos. Para tal, construíram-se duas pequenas aplicações de teste: uma delas faz o registo de um utilizador numa sessão cooperativa e aguarda a notificação de eventos; a outra faz o registo de um utilizador na mesma sessão e produz eventos cooperativos a cadências pré-determinadas. Na aplicação que produz os eventos

regista-se o tempo imediatamente anterior ao envio de um evento e na aplicação receptora regista-se o tempo imediatamente posterior à sua recepção. A diferença entre os dois tempos registados permite obter um valor aproximado do atraso que a utilização da notificação de eventos introduz no desempenho das aplicações cooperativas.

O registo dos instantes de produção e recepção dos eventos cooperativos coloca um problema de difícil resolução: estando as aplicações situadas em computadores distintos, as medições só são fiáveis se os relógios dos dois computadores estiverem perfeitamente sincronizados ou, pelo menos, a diferença entre ambos não for suficiente para afectar a precisão dos valores obtidos. Numa primeira abordagem, efectuaram-se medições com as aplicações localizadas em diferentes computadores, fazendo-se a sincronização dos seus relógios recorrendo a ferramentas específicas para o efeito. De facto, existem ferramentas que permitem acertar o relógio de vários computadores a partir de um outro. Contudo, a utilização de uma dessas ferramentas [171] não se revelou satisfatória, pois a diferenças entre os relógios de dois computadores atingiam frequentemente as várias centésimas de milissegundos, o que fez com que chegassem a ocorrer valores negativos na diferença entre os instantes de recepção e produção de eventos.

A dificuldade em obter medições temporais fiáveis com as aplicações localizadas em computadores distintos, motivou a procura de uma solução em que ambas as aplicações se encontrassem no mesmo computador. Neste caso, a base temporal é a mesma, mas não é medido o tempo gasto com a transmissão das mensagens na rede. No entanto, este valor não é condicionado pelo ambiente cooperativo que foi implementado, sendo uma componente temporal que existe em qualquer situação que envolva a transmissão de mensagens na rede. Contudo, o seu valor continua a afectar o desempenho global das aplicações, pois estas têm necessariamente que transmitir mensagens entre si. De facto, se a soma dos valores dos tempos de transmissão e dos tempos imputáveis ao ambiente cooperativo for significativo, pode tornar as aplicações cooperativas pouco atractivas. Porém, os atrasos típicos introduzidos pelo processamento das mensagens IP e pela transmissão de pacotes numa rede local são da ordem das dezenas ou centenas de milissegundos; se os tempos medidos pelas aplicações de teste forem da mesma ordem de grandeza, então o desempenho do sistema poderá ser considerado satisfatório.

Com base na validade destes pressupostos, efectuaram-se vários testes em que se produziram 200 eventos a uma cadência constante, variando essa cadência para cada um dos testes. Assim, fizeram-se testes em que o intervalo decorrido entre a produção de dois eventos consecutivos variou entre 0,1 s e 10 s. Os resultados obtidos nesses testes encontram-se resumidos na Tabela 5-1.

	Intervalo entre eventos consecutivos (s)					
	0,1	0,5	1	2	5	10
Média	49,498	0,326	0,073	0,059	0,055	0,061
Máximo	111,701	2,433	0,320	0,281	0,241	0,331
Mínimo	1,713	0	0	0	0	0

Tabela 5-1 – Resultados dos testes de desempemho

A análise dos valores da tabela revela que para um intervalo de produção de eventos de 0,1 s o sistema exibe um fraco desempenho, com atrasos crescentes, com um tempo médio de atraso de mais de 49 s e com um valor máximo que atinge quase 2 minutos. Observando o gráfico da Figura 5-28, que mostra a evolução dos atrasos ao longo da simulação, verifica-se que o sistema não foi capaz de suportar uma cadência de produção de eventos tão elevada, acumulando os atrasos sucessivos. Esta é, porém, uma situação de carga muito elevada, não sendo vulgar a sua ocorrência num cenário real de trabalho cooperativo.

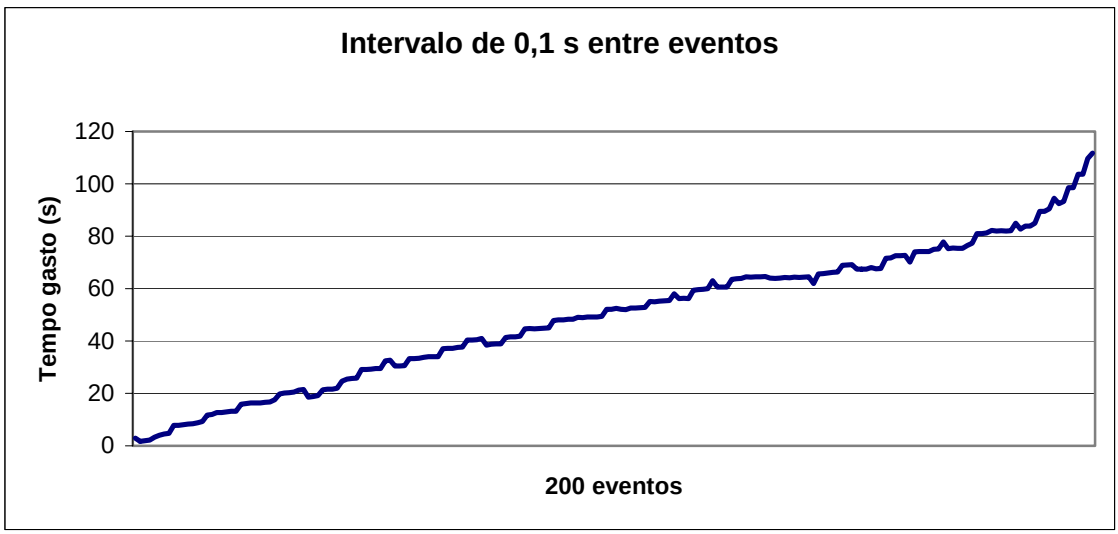

Figura 5-28 – Gráfico de desempenho para um intervalo entre eventos de 0,1 s

Para intervalos entre eventos de 0,5 s o valor médio do atraso já é aceitável, apesar de ainda existirem alguns valores de pico. Contudo, a esmagadora maioria dos atrasos situa-se na casa das dezenas ou algumas centenas de milissegundos, como se pode ver no gráfico da Figura 5-29, valores esses que se podem considerar perfeitamente aceitáveis. Note-se ainda que o valor mínimo de atraso registado foi nulo. Tal facto ocorre porque o método Java que permite obter o instante actual, `java.util.Date.getTime()`, não consegue registar tempos inferiores a 1 ms. Assim, pode-se concluir que o atraso sofrido é, neste caso, de uma ordem de grandeza inferior aos milissegundos.

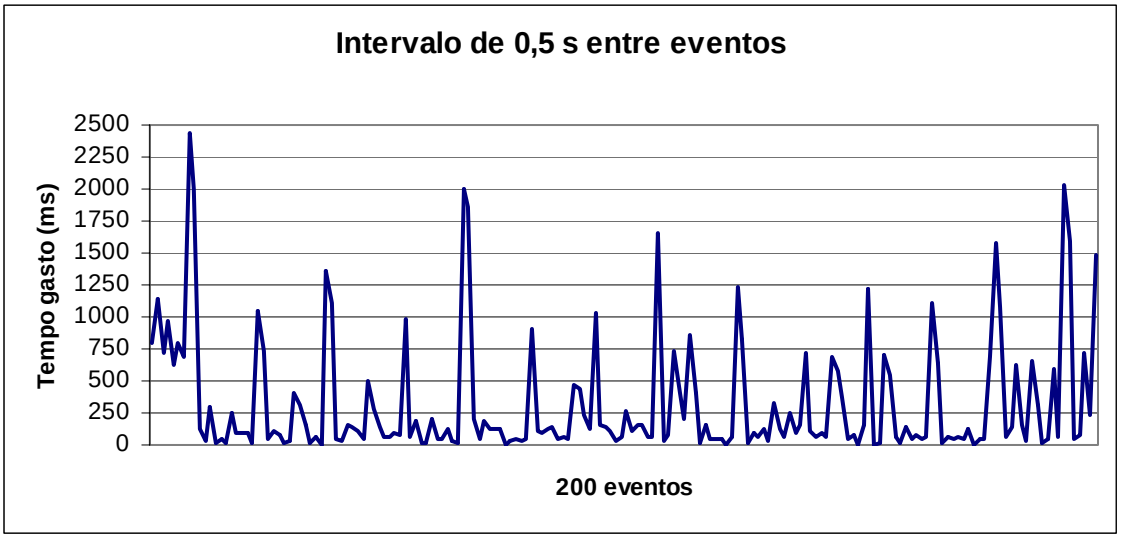

Figura 5-29 - Gráfico de desempenho para um intervalo entre eventos de 0,5 s

Os resultados obtidos para intervalos entre eventos a partir de 1 s mostram valores médios estabilizados e da ordem das décimas de segundo, com valores máximos na casa das centésimas de segundo. Mais uma vez, os valores mínimos registados são nulos, concluindo-se que estarão na casa dos microssegundos ou menos. Os gráficos que representam os resultados obtidos para intervalos entre eventos de 1 s, 2 s, 5 s e 10 s podem ser vistos na Figura 5-30, onde se pode observar a grande semelhança existente entre eles.

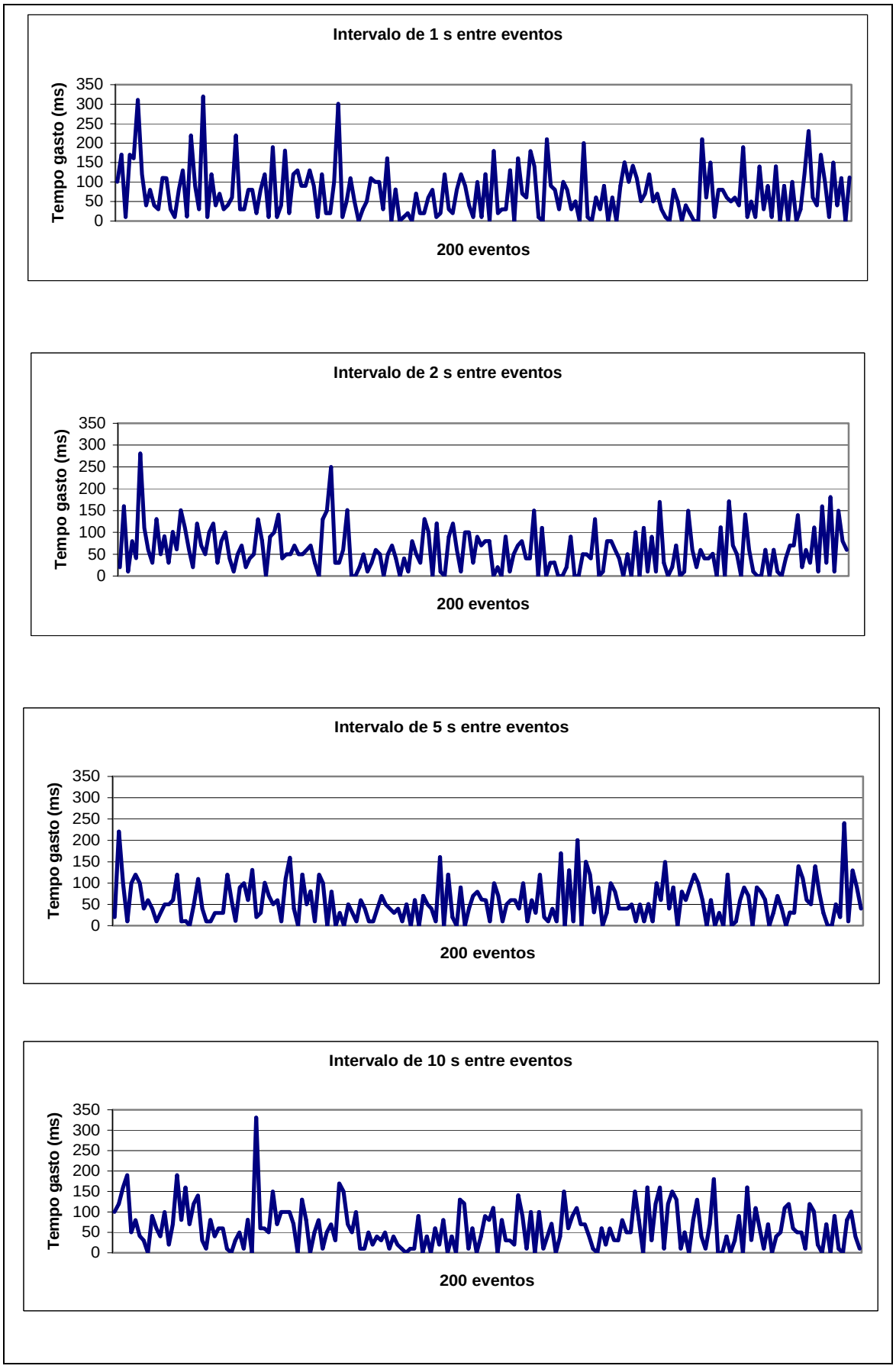

Figura 5-30 – Gráficos de desempenho para intervalos entre eventos de 1 a 10 s

Verifica-se, assim, existir um bom desempenho do sistema para cadências de produção de eventos semelhantes às que serão esperadas numa situação real. Os atrasos médios e máximos relativos ao sistema de propagação de eventos, somados aos atrasos usuais de transmissão de pacotes em redes locais, produzem valores globais abaixo de 1 s, os quais, por não serem perceptíveis, não interferem com o desenrolar das actividades cooperativas.

Capítulo 6

Conclusões

Concluído o desenvolvimento e a avaliação, importa fazer uma análise crítica das características do modelo proposto e dos resultados obtidos com o ambiente de criação de aplicações cooperativas que foi implementado e tirar conclusões acerca da forma como os objectivos definidos inicialmente foram ou não atingidos. Por fim, faz-se uma perspectivação da evolução futura do trabalho.

6.1. Conclusões

O propósito principal desta Tese de Doutoramento foi o de definir e testar um modelo de criação de serviços cooperativos que utilizasse tecnologias *web*, formando uma base de trabalho para a implementação dos mais variados tipos de aplicações cooperativas de uma forma modular e interoperável. De forma a testar o modelo proposto, desenvolveu-se um ambiente de criação de aplicações cooperativas, que implementa os principais serviços necessários para a construção e execução de aplicações cooperativas. Para testar o ambiente cooperativo, construiu-se um protótipo de uma aplicação de edição cooperativa de vídeo, o qual permitiu verificar o desempenho do sistema com uma aplicação que comporta alguma complexidade.

Para a concretização dos objectivos traçados cumpriram-se as seguintes etapas:

- foram estudados os aspectos mais importantes dos sistemas distribuídos e as principais arquitecturas de processamento distribuído, analisando as suas características, as vantagens que oferecem e os problemas que colocam;
- foram estudados variados aspectos relacionados com o trabalho cooperativo suportado por computador e com a classificação das aplicações cooperativas, analisando os seus requisitos e as suas principais condicionantes;
- fez-se uma recolha de informação relativa aos sistemas cooperativos que têm sido desenvolvidos, para melhor aferir o seu estado de evolução e identificar áreas onde não tenha sido desenvolvido trabalho significativo;
- propôs-se um modelo para a criação de serviços cooperativos concordante com os propósitos da Tese;
- pesquisaram-se implementações das tecnologias escolhidas para a implementação do modelo, nomeadamente as tecnologias de *web services*, de bases de dados de objectos, de manipulação de informação audiovisual e de propagação de eventos;
- como resultado dessa pesquisa, escolheram-se as tecnologias que melhor serviam os objectivos que se pretendiam atingir, dentro dos limites

definidos pelas condicionantes económicas, temporais e da adequação ao ambiente computacional existente, sem provocar rupturas que exigissem um esforço de adaptação que não se revelasse compensatório;

- implementaram-se protótipos dos serviços e sistemas de informação definidos no modelo, para os quais, nalguns casos, não foram implementadas todas as suas funcionalidades, mas apenas aquelas que se julgou serem importantes para a validação do modelo no contexto da aplicação escolhida como caso de estudo;
- resultando da recolha de informação relativa às aplicações cooperativas que são conhecidas, escolheu-se a edição cooperativa de vídeo para implementar um protótipo de uma aplicação inovadora que permitisse validar o modelo proposto;
- implementou-se um protótipo de uma aplicação simplificada de edição de vídeo cooperativo, para testar o funcionamento dos serviços implementados e para verificar a adequação destes a uma aplicação com alguma complexidade tecnológica;
- realizaram-se testes ao funcionamento dos serviços e das aplicações numa rede local, analisando a correcção da execução dos vários componentes e avaliando qualitativamente o desempenho do sistema em termos de tempo de resposta às acções desencadeadas pelos utilizadores.

Os objectivos definidos no capítulo de introdução foram, na generalidade, atingidos com sucesso, apresentando contribuições inovadoras para a área de conhecimento em que se integra o trabalho desenvolvido nesta tese. O modelo de criação de serviços cooperativos revelou características de abertura, distribuição, interoperabilidade, modularidade e capacidade de actualização e de evolução. O ambiente de criação de aplicações cooperativas que foi construído com base no modelo possui funcionalidades suficientemente genéricas para se adaptar a variados tipos de aplicações. A sua utilização para o caso da edição cooperativa de vídeo mostrou-se viável e o desempenho exibido foi encorajador.

A utilização dos *web services* como tecnologia de suporte a sistemas cooperativos revelou-se viável e de eficácia satisfatória, possibilitando a construção de serviços que permitem implementar funcionalidades básicas, a partir das quais se podem construir diversas aplicações cooperativas. Os serviços implementados revelaram capacidade para suportar uma cadência de invocações apreciável, sem prejuízo visível do desempenho das aplicações. Somente para cargas elevadas se notou algum aumento do tempo de propagação das notificações de eventos, correspondendo, no entanto, a situações pouco esperadas numa utilização normal.

Outro componente importante do modelo é o que diz respeito ao armazenamento de informação relativa aos utilizadores, às aplicações e à informação multimédia em bases de dados de objectos. A utilização destas mostrou-se satisfatória, permitindo o armazenamento persistente de dados recorrendo a um modelo de objectos compatível com o usado no resto da implementação. O recurso a *blobs* para o armazenamento da informação tornou mais transparente o acesso a esta, não sendo necessário que os utilizadores se preocupem com a sua localização. As procuras de informação nas bases de dados introduziram uma componente temporal pouco sensível e que, de qualquer forma, é habitual nas pesquisas em bases de dados.

A solução encontrada para a notificação dos eventos ocorridos nas aplicações permitiu separar o serviço respectivo da tecnologia que proporciona a sua distribuição, sendo essa abordagem também utilizada nos outros serviços implementados. A tecnologia de distribuição de eventos utilizada (JSDT) revelou um funcionamento simples e de fácil programação, com um desempenho robusto nas situações testadas.

O componente que se mostrou menos satisfatório, em termos de facilidade de programação e do desempenho revelado, foi o responsável pela manipulação de informação audiovisual (JMF). De facto, esta tecnologia mostrou alguma complexidade em termos da eficaz utilização das suas capacidades, nomeadamente no que diz respeito à resposta da sua máquina de estados, que nem sempre se revelou satisfatória. Tal facto pode estar relacionado com uma incorrecta percepção e utilização da tecnologia, mas isso também resulta das falhas de transparência e simplicidade que ela apresenta ao programador. Para além disso, o seu tempo de resposta também pareceu algo elevado. Porém, estas aparentes deficiências não invalidam a sua utilização como ferramenta de suporte à construção de protótipos de aplicações multimédia, permitindo a manipulação

de diversos formatos audiovisuais sem necessidade de construção de módulos codificadores e descodificadores.

O facto dos serviços que foram construídos apresentarem uma implementação bastante genérica, deixando as especificidades entregues a aplicações de suporte, confere-lhes capacidade de evolução e independência tecnológica. Este facto permite, de alguma forma, extrapolar os resultados obtidos no protótipo implementado para outros tipos de aplicações cooperativas, o que, contudo, não valida totalmente o modelo para todas as situações, as quais necessitarão sempre de ser testadas.

O protótipo da aplicação cooperativa de edição de vídeo foi implementado de uma forma muito simplificada, dada a grande complexidade que comporta o desenvolvimento completo de uma aplicação deste tipo. As simplificações feitas não impediram, porém, que a aplicação apresentasse as características suficientes para testar a validade do modelo de serviços cooperativos e para demonstrar que é possível construir uma aplicação deste tipo recorrendo às tecnologias seleccionadas.

Foram realizados testes ao funcionamento do ambiente cooperativo e ao seu desempenho. Pretendeu-se, essencialmente, verificar se os serviços e as aplicações implementadas funcionavam correctamente e se eram capazes de dar resposta a situações diversas de carga, no que diz respeito à cadência de produção de eventos cooperativos. Para tal, testou-se o ambiente em situações reais de utilização da aplicação de edição de vídeo em modo cooperativo, não se tendo verificado falhas no seu funcionamento, nem atrasos consideráveis na propagação dos eventos. Para testar o sistema em situações de carga mais elevada, construíram-se pequenos programas que simularam o registo dos utilizadores em sessões cooperativas e a produção massiva de eventos. Verificou-se que o desempenho se manteve satisfatório, sendo mais sensíveis os atrasos sofridos na propagação dos eventos para cadências a partir de dois eventos por segundo. Note-se, contudo, que esta avaliação é essencialmente qualitativa, dada a dificuldade existente na obtenção de medições fiáveis em ambientes distribuídos.

6.2. Perspectivas de evolução

O modelo proposto nesta tese não pretende ser a solução definitiva para o desenvolvimento de serviços cooperativos, sendo antes uma proposta que poderá servir satisfatoriamente o problema em questão e fomentar a sua discussão e o desenvolvimento de outras soluções.

Os serviços cooperativos que foram construídos não implementam a totalidade das funcionalidades contempladas no modelo proposto, necessitando de trabalho futuro neste campo. Para além disso, apenas se implementou uma única aplicação cooperativa, representativa de uma pequena parcela do universo de categorias de aplicações cooperativas identificadas. A própria escolha das acções na aplicação que produzem eventos cooperativos não foi sujeita a um estudo prévio acerca das necessidades reais dos utilizadores habituais deste tipo de aplicações, sendo esta uma área que pode futuramente beneficiar da integração de uma equipa multidisciplinar.

A implementação de outras aplicações cooperativas, representativas das diversas categorias existentes, pode ajudar a uma validação mais completa do modelo e, certamente, ao seu aperfeiçoamento. Um dos serviços propostos no modelo, o Serviço de Controlo de Concorrência, não foi sequer implementado, por não ser necessário utilizá-lo com o protótipo de aplicação de edição cooperativa de vídeo. Contudo, este serviço pode revelar-se fundamental em categorias de aplicações cooperativas que sejam mais exigentes quanto à disponibilização de mecanismos de garantia de consistência da informação partilhada.

A localização da informação de que as aplicações necessitam foi abordada de uma forma simples, sendo efectuadas pesquisas por nome e por palavras-chave. No entanto, podem-se efectuar procuras mais elaboradas, usando outros critérios de pesquisa ou combinando vários critérios. A própria utilização de metadados deve ser aperfeiçoada, usando um formato de representação de metadados, como o RDF [159], o Dublin Core [160] ou o MPEG-7 [161].

Os protótipos implementados utilizam os *web services* através da chamada de procedimentos remotos, não tirando partido da possibilidade de usar directamente a troca de mensagens XML. A utilização destas é outra área onde futuramente se pode produzir algum trabalho.

Os serviços construídos no ambiente cooperativo não implementam mecanismos de segurança, apesar da infra-estrutura de *web services* o permitir e de este ser um aspecto com crescente importância. Por exemplo, o Serviço de Autenticação seria mais seguro se encriptasse os dados do utilizador. Assim, evoluções futuras dos serviços e das aplicações apresentadas nesta tese deverão incluir mecanismos de segurança.

Uma área onde os resultados obtidos nem sempre foram tão positivos quanto desejável foi a da manipulação de informação multimédia, a qual é, contudo, muito importante no contexto das aplicações cooperativas. Interessa, assim, aperfeiçoar esse componente do sistema ou encontrar soluções tecnológicas alternativas.

A forma como acedemos às aplicações vai mudando com o tempo e assiste-se, neste momento, a um despertar e à previsível consolidação da utilização de dispositivos móveis para acesso a aplicações baseadas na *web*. Os telefones móveis mais recentes, principalmente os da 3ª geração, já permitem o acesso *web* a algumas aplicações mais ou menos sofisticadas, essencialmente de entretenimento. Os operadores móveis disponibilizam igualmente acessos de 3ª geração para computadores portáteis, oferecendo-lhes, assim, acesso de banda larga à Internet. Existem diversos dispositivos, como as agendas electrónicas, que possuem ligações sem fios que lhes permitem ligar-se a redes locais ou funcionar também como telefone móvel. Os computadores portáteis com ligações sem fios assumem igualmente uma importância crescente, sendo já relativamente comum a existência de redes sem fios nas organizações e em locais públicos de grande movimento. Esta variedade de dispositivos móveis de variadas características a par dos acessos fixos convencionais, introduz um novo tipo de heterogeneidade, que passa a existir não só ao nível dos ambientes computacionais, mas também ao nível das formas de visualização da informação (diversas capacidades gráficas) e da interacção com o dispositivo (ratos, teclados, dispositivos de toque). Tal heterogeneidade é bastante provável numa organização que utilize aplicações cooperativas, facto que deve motivar a investigação de formas de personalização da informação e das interfaces gráficas em função dos dispositivos utilizados.

Referências Bibliográficas

[1] Tanenbaum, A.S., *Modern Operating Systems - Part 2: Distributed Operating Systems*. 1992, New Jersey: Prentice-Hall International.

[2] Marques, J.A. and P. Guedes, *Tecnologia de Sistemas Distribuídos*. Tecnologias de Informação. 1998: FCA.

[3] Comer, D.E. and D.L. Stevens, *Internetworking with TCP/IP Volume III - Client/Server Programming and Applications*. 1994, New Jersey: Prentice-Hall International.

[4] Birrel, A. and B. Nelson, *Implementing Remote Procedure Calls*, in *ACM Transactions on Computer Systems*. 1984. p. 39-59,

[5] Constantinescu, Z. and P. Petrovic, *Q2ADPZ* An Open Source, Multi-platform System for Distributed Computing*, in *ACM Crossroads*. 2002. p. 13-20,

[6] SETI@home, http://setiathome.ssl.berkeley.edu

[7] Borghoff, U.M. and J.H. Schlichter, *Producer-consumer interaction*, in *Computer-Supported Cooperative Work*. 1998, Springer-Verlag. p. 18-19.

[8] Ben-Ari, M., *Principles of Concurrent and Distributed Programming*. Series in Computer Science, ed. C.A.R. Hoare. 1990, New York: Prentice-Hall International.

[9] Liebig, C., et al. *A publish/subscribe CORBA Persistent State Service Prototype*. in *Middleware 2000*. 2000: Springer-Verlag.

[10] Opyrchal, L., et al. *Exploiting IP Multicast in Content-Based Publish-Subscribe Systems*. in *Middleware 2000*. 2000: Springer-Verlag.

[11] DCOM Specification, http://www.microsoft.com/TechNet/prodtechno1/winntas/plan/dcomtowp.asp

[12] Nina, N., *DLL*, in *Visual Basic 6 - Curso Completo*. 1999, FCA. p. 599-600.

[13] COM+, http://www.chappellassoc.com/articles/article_COM.html

[14] COM+, http://www.execpc.com/~gopalan/com/complus.html

[15] INFO: What's New with COM+ 1.0 (Q253669), http://support.microsoft.com/default.aspx?scid=kb;EN-US;q253669

[16] Sun, *Java Remote Method Invocation Specification*. 1999, Sun Microsystems,

[17] Cockayne, W.T. and M. Zyda, *Mobile Agents*. 1998: Manning Publications Co.

[18] Pham, V.A. and A. Karmouch, *Mobile Software Agents: An Overview*, in *IEEE Communications Magazine*. 1998. p. 26-37,

[19] Hamilton, G., R. Cattell, and M. Fisher, *JDBC Database Access with Java: A Tutorial and Annotated Reference*. 1997: Addison-Wesley Pub. Co.

[20] Java Native Interface Specification, http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/jniTOC.doc.html

[21] OMG, *The Common Object Request Broker: Architecture and Specification*. 1991, OMG,

[22] Vinoski, S., *Distributed Object Computing with CORBA*, in *C++ Report Magazine*. 1993,

[23] Vinoski, S., *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments*, in *IEEE Communications Magazine*. 1997,

[24] OMG, *The Common Object Request Broker: Architecture and Specification*. 1995, OMG,

[25] OMG, *Interim Report from CORBA Core 2002 RTF*. 2002, OMG,

[26] What's Coming in CORBA 3, http://www.omg.org/technology/corba/corba3releaseinfo.htm

[27] Vinoski, S., *New Features for CORBA 3.0*, in *Communications of the ACM*. 1998,

[28] OMG, *Realtime CORBA*. 1998, OMG,

[29] OMG, *The Common Object Request Broker: Architecture and Specification*. 2001, OMG,

[30] Schmidt, D., *A High-Performance End System Architecture for Real-time CORBA*, in *IEEE Communications Magazine*. 1997,

[31] An Overview of the Joint Real-time CORBA submission, http://www.cs.wustl.edu/~schmidt/PDF/RT-CORBA.pdf

[32] Real-time CORBA, http://www.cs.wustl.edu/~schmidt/PDF/rtcorba4.pdf

[33] Nagappan, R., R. Skoczylas, and R.P. Sriganesh, *Developing Java Web Services*. 2003: Wiley.

[34] W3C, http://www.w3c.org/

[35] Web Services, http://www.webservices.org/

[36] HTTP, http://www.w3.org/Protocols/

[37] RFC 2616: HTTP/1.1, ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt

[38] XML, http://www.xml.org/

[39] WSDL, http://www.w3.org/TR/wsdl

[40] SOAP, http://www.w3.org/TR/soap/

[41] UDDI, http://www.uddi.org/

[42] SOAP 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[43] SOAP 1.2, http://www.w3.org/TR/soap12-part1/

[44] RFC 821: Simple Mail Transfer Protocol, ftp://ftp.rfc-editor.org/in-notes/rfc821.txt

[45] W3C XML Schema Part 2: Datatypes Recommendation, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[46] SOAP Messages with Attachments, http://www.w3.org/TR/SOAP-attachments

[47] ebXML, http://www.ebxml.org/

[48] Sun ONE, http://wwws.sun.com/software/sunone/

[49] Microsoft .NET Framework, http://msdn.microsoft.com/netframework/

[50] IBM Websphere, http://www-306.ibm.com/software/info1/websphere/index.jsp

[51] Systinet WASP, http://www.systinet.com/

[52] BEA WebLogic, http://dev2dev.bea.com

[53] Roque, V.M.G. and J.L. Oliveira, *CORBA, DCOM e JavaRMI - Uma análise comparativa*, in *Ingenium*. 2000. p. 87-91,

[54] A Detailed Comparison of CORBA, DCOM and Java/RMI, http://www.execpc.com/~gopalan/misc/compare.html

[55] DCOM and CORBA Side by Side, Step by Step, and Layer by Layer, http://www.bell-labs.com/~emerald/dcom_corba/Paper.html

[56] Borghoff, U.M. and J.H. Schlichter, *Computer-Supported Cooperative Work*. 1998: Springer-Verlag. 529.

[57] Beaudouin-Lafon, M., et al., *Computer Supported Co-operative Work*. Trends in Software, ed. B. Krishnamurthy. 1999: John Wiley & Sons. 258.

[58] Grudin, J., *CSCW: History and Focus*. IEEE Computer, 1994. **27**(5): p. 19-26.

[59] Bradner, E. and G. Mark. *Social Presence with Video and Application Sharing*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[60] Bulmer, L. and P. Dew. *A Study of Collaboration Using Jigsaw Puzzles*. in *CVE'02*. 2002. Bona, Alemanha: ACM.

[61] Fussell, S.R., R.E. Kraut, and J. Siegel. *Coordination of Communication: Effects of Shared Visual Context on Collaborative Work*. in *CSCW'00*. 2000. Philadelphia, USA: ACM.

[62] Hudson, T., et al. *Enabling Distributed Collaborative Science*. in *CSCW'00*. 2000. Philadelphia, USA: ACM.

[63] Yamauchi, Y., et al. *Collaboration with Lean Media: How Open-Source Software Succeeds*. in *CSCW'00*. 2000. Philadelphia, USA: ACM.

[64] ICQ, http://www.icq.com/

[65] Milner, R., *Communication and Concurrency*. 1995, Englewood Cliffs, USA: Prentice-Hall.

[66] Mageel, J. and J. Kramer, *Concurrency: State Models and Java Programs*. 1999, New York, USA: John Wiley & Sons.

[67] Helal, A.S., A.A. Heddaya, and B.B. Bhargava, *Replication Techniques in Distributed Systems*. Series on Advances in Database Systems. 1996: Dordrecht: Kluwer.

[68] Moser, L.E., V. Kapur, and P.M. Melliar-Smith. *Probabilistic Language Analysis of Weighted Voting Algorithms*. in *ACM SIGMETRICS*. 1990. Boulder, USA: ACM.

[69] Garcia-Molina, H. and D. Barbara. *Optimizing the Reliability Provided by Voting Mechanisms*. in *4th IEEE Conf. on Distributed Computing Systems*. 1984. São Francisco, USA: IEEE Computer Society Press.

[70] Lynch, N., et al., *Atomic Transactions*. 1994, San Mateo, USA: Morgan Kaufmann.

[71] NetMeeting, http://www.microsoft.com/NetMeeting/

[72] WfMC, http://www.wfmc.org/

[73] Hofte, G.H.t., et al. *Exploring a Design Space for Place-based Presence*. in *CVE'02*. 2002. Bona, Alemanha: ACM.

[74] Mackay, W., *Media Spaces: Environments for Informal Multimedia Interaction*, in *Computer Supported Co-operative Work*, M. Beaudouin-Lafon, Editor. 1999, John Wiley & Sons. p. 55-82.

[75] Nichols, D.M. and M.B. Twidale. *Matchmaking and privacy in the Digital Library: striking the right balance*. in *4th UK/International Conference on Electronic Library and Visual Information Research (ELVIRA 4)*. 1997. London, UK.

[76] Ackerman, M.S. and D.W. McDonald. *Answer Garden 2: Merging Organizational Memory with Collaborative Help*. in *CSCW'96*. 1996. Cambridge, USA.

[77] Grather, W. and W. Prinz. *The Social Web Cockpit: Support for Virtual Communities*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[78] Agentware, http://www.autonomy.com

[79] Yankelovich, N., J.B. Begole, and J.C. Tang. *Sun™ SharedShell Tool*. in *CSCW'00*. 2000. Philadelphia, USA: ACM.

[80] Antunes, P. and N. Guimarães. *Transforming Formal and Informal Work Processes*. in *CRIWG'97*. 1997. San Lorenzo de El Escorial.

[81] Petrie, C. and S. Sarin, *Beyond Documents: Sharing Work*, in *IEEE Internet Computing*. 2000. p. 34-36,

[82] Maurer, F., et al., *Merging Project Planning and Web-Enabled Dynamic Workflow Technologies*, in *IEEE Internet Computing*. 2000. p. 65-74,

[83] Lotus Notes, http://www.lotus.com/

[84] Microsoft Exchange, http://www.microsoft.com/exchange/default.asp

[85] Munkvold, B.E. and R. Anson. *Organizational Adoption and Diffusion of Electronic Meeting Systems: A Case Study*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[86] Bellassai, G., et al. *SISCO: A tool to improve meetings productivity*. in *CRIWG'95*. 1995. Lisboa.

[87] Cavalcanti, M.C., M.R.S. Borges, and M.Y. Endo. *SISCO-RIO: An Asynchronous System to Support Meeting Preparation*. in *CRIWG'97*. 1997. San Lorenzo de El Escorial.

[88] Espinosa, J., J.A. Pino, and P. Pollard. *On the development of a SISCO implementation using Java*. in *CRIWG'97*. 1997. San Lorenzo de El Escorial.

[89] Antunes, P. and N. Guimarães. *NGTool - Exploring Mechanisms of Support to Interactivity in the Group Process*. in *CRIWG'95*. 1995. Lisboa.

[90] Wiberg, M. *RoamWare: An Integrated Architecture for Seamless Interaction In between Mobile Meetings*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[91] Li, D. and J. Patrao. *Demonstrational Customization of a Shared Whiteboard to Support User-Defined Semantic Relationships among Objects*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[92] Ruiz, D.C. and J. Favela. *Collaborative Review and Edition of HTML Documents*. 1998.

[93] Yang, Y., et al., *Real-Time Cooperative Editing on the Internet*, in *IEEE Internet Computing*. 2000. p. 18-25,

[94] Jr., U.F. and J.-M. Farines. *A Support Platform for Distributed Editing*. in *CRIWG'95*. 1995. Lisboa.

[95] Knister, M.J. and A. Prakash. *DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors*. in *CSCW'90*. 1990. Los Angeles, USA: ACM.

[96] Decouchant, D., M.R. Salcedo, and M. Serrano. *Design Issues of A Cooperative Authoring Application*. in *CRIWG'97*. 1997. San Lorenzo de El Escorial.

[97] Sun, C. *Undo Any Operation at Any Time in Group Editors*. in *CSCW'00*. 2000. Philadelphia, USA: ACM.

[98] González, O.M., et al. *PENCACOLAS "PEN Computer Aided Composing cOLlAborative System" Groupware for Learning*. in *CRIWG'97*. 1997. San Lorenzo de El Escorial.

[99] Caglayan, A. and C. Harrison, *AGENT Sourcebook: A Complete Guide to Desktop, Internet and Intranet Agents*. 1997: John Wiley & Sons, Inc.

[100] Vidal, J.M., P.A. Buhler, and M.N. Huhns, *Inside an Agent*, in *IEEE Internet Computing*. 2001. p. 82-86,

[101] Wainer, J. and C. Ellis. *Agents in Groupware Systems*. 1999.

[102] Bond, A.H. and L. Gasser, *Readings in Distributed Artificial Intelligence*. 1988, San Mateo, USA: Morgan Kaufmann.

[103] Russell, S. and P. Norvig, *Artificial Intelligence: A Modern Approach*. 1995, Englewood Cliffs, USA: Prentice-Hall.

[104] ARPA, http://www.arpa.gov

[105] Debnath, S., S. Sen, and B. Blackstock, *LawBot: A Multiagent Assistant for Legal Research*, in *IEEE Internet Computing*. 2000. p. 32-37,

[106] Wainer, J. and D.P. Braga. *Symgroup: Applying Social Agents in a Group Interaction System*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[107] Sarwar, B.M., et al. *Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System*. in *CSCW'98*. 1998. Seattle, USA: ACM.

[108] Collis, B., *Applications of Computer Communications in Education: An Overview*, in *IEEE Communications Magazine*. 1999. p. 82-86,

[109] Ausserhofer, A., *Web-Based Teaching and Learning: A Panacea?*, in *IEEE Communications Magazine*. 1999. p. 92-96,

[110] Azuma, J., *Creating Educational Web Sites*, in *IEEE Communications Magazine*. 1999. p. 109-113,

[111] Harris, D.A., *Online Distance Education in the United States*, in *IEEE Communications Magazine*. 1999. p. 87-91,

[112] Roschelle, J., et al., *Developing Educational Software Components*, in *IEEE Computer*. 1999. p. 50-58,

[113] Guerrero, L.A. and D.A. Fuller. *CLASS: A Computer Platform for the Development of Education's Collaborative Applications*. in *CRIWG'97*. 1997. San Lorenzo de El Escorial.

[114] Harasim, L., *A Framework for Online Learning: The Virtual-U*, in *IEEE Computer*. 1999. p. 44-49,

[115] Marquès, J.M. and L. Navarro. *WWG: a Wide-Area Infrastructure to Support Groups*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[116] Núñez, G., U. Aguero, and C. Olivares. *Group Decision-Making for Collaborative Educational Games*. 1998.

[117] Pinto, J.S. and J.A. Martins. *Design and Implementation of a Virtual Learning Environment*. in *CRIWG'95*. 1995. Lisboa.

[118] Regan, P.M. and B.M. Slator. *Case-based Tutoring in Virtual Education Environments*. in *CVE'02*. 2002. Bona, Alemanha: ACM.

[119] Conti, G., G. Ucelli, and J. Petric. *JCAD-VR: a Collaborative Design Tool for Architects*. in *CVE'02*. 2002. Bona, Alemanha: ACM.

[120] Kauff, P. and O. Schreer. *An Immersive 3D Video-Conferencing System Using Shared Virtual Team User Environments*. in *CVE'02*. 2002. Bona, Alemanha: ACM.

[121] MacColl, I., et al. *Shared Visiting in EQUATOR City*. in *CVE'02*. 2002. Bona, Alemanha: ACM.

[122] Brave, S., H. Ishii, and A. Dahley. *Tangible Interfaces for Remote Collaboration and Communication*. in *CSCS'98*. 1998. Seattle, USA: ACM.

[123] Araujo, R.M.d. and H. Fuks. *QUORUM-W: A Group Decision Support Tool for the Internet Environment*. in *CRIWG'95*. 1995. Lisboa.

[124] Kim, Y., S.-H. Kang, and D. Kim, *WW-FLOW: Web-Based Workflow Management with Runtime Encapsulation*, in *IEEE Internet Computing*. 2000. p. 55-64,

[125] Bergholz, A., *Extending your Markup: An XML Tutorial*, in *IEEE Internet Computing*. 2000. p. 74-79,

[126] Hayes, J.G., et al., *Workflow Interoperability Standards for the Internet*, in *IEEE Internet Computing*. 2000. p. 37-45,

[127] Li, S.F., Q. Stafford-Fraser, and A. Hopper, *Integrating Synchronous and Asynchronous Collaboration with Virtual Network Computing*, in *IEEE Internet Computing*. 2000. p. 26-33,

[128] Cadiz, J.J., A. Balachandran, and E. Sanocki. *Distance Learning Through Distributed Collaborative Video Viewing*. in *CSCW'00*. 2000. Philadelphia, USA: ACM.

[129] Costa, J.M., et al. *Videoconferência Multiponto*. in *CRC'98*. 1998. Coimbra.

[130] Hoshi, T., et al., *B-ISDN Multimedia Communication and Collaboration Platform Using Advanced Video Workstations to Support Cooperative Work*, in *IEEE Journal on Selected Areas in Communications*. 1992. p. 1403-1412,

[131] Martínez, J.A.A., et al. *An Environment for Supporting Interactive Presentations to Distributed Audiences over the World-Wide Web*. in *CRIWG'97*. 1997. San Lorenzo de El Escorial.

[132] Baresi, L., et al. *WIDE Workflow Development Methodology*. in *WACC'99*. 1999. São Francisco, USA: ACM.

[133] Bolcer, G.A., *Magi: An Architecture for Mobile and Disconnected Workflow*, in *IEEE Internet Computing*. 2000. p. 46-54,

[134] Tolksdorf, R., *Workspaces: A Web-Based Workflow Management System*, in *IEEE Internet Computing*. 2002. p. 18-26,

[135] Weske, M., et al. *A Reference Model for Workflow Application Development Processes*. in *WACC'99*. 1999. São Francisco, USA: ACM.

[136] Geyer, W., et al. *A Team Collaboration Space Supporting Capture and Access of Virtual Meetings*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[137] Chen, D. and C. Sun. *Undoing Any Operation in Collaborative Graphics Editing Systems*. in *GROUP'01*. 2001. Boulder, USA: ACM.

[138] Oliveira, P.C.R.C., *Um Modelo baseado em Agentes Móveis para Procura de Informação em Sistemas de Informação Distribuídos*. 2003, UTAD: Vila Real,

[139] Chatzipapadopoulos, F.G., M.K. Perdikeas, and I.S. Venieris, *Mobile Agent and CORBA Technologies in the Broadband Intelligent Network*, in *IEEE Communications Magazine*. 2000. p. 116-124,

[140] Becker, K. and A.L. Zanella, *A Cooperation Model for Teaching/Learning Modeling Disciplines*. 1999,

[141] Carver, C., *Building a Virtual Community for a Tele-Learning Environment*, in *IEEE Communications Magazine*. 1999. p. 114-118,

[142] Takefuji, Y., et al., *ATM and Wireless Experiments for Remote Lectures*, in *IEEE Communications Magazine*. 1999. p. 98-101,

[143] Collaborative Visualization of Large Scale Hypermedia Databases, http://www.crg.cs.nott.ac.uk/~ccb/papers/GMD-paper.html

[144] Matsuda, K., T. Miyake, and H. Kawai. *Culture Formation and its Issues in Personal Agent-oriented Virtual Society - "PAW^2"*. in *CVE'02*. 2002. Bona, Alemanha: ACM.

[145] Nam, T.-J. *An investigation of multi-user design tools for collaborative 3-D modeling*. in *CSCW'98 Doctoral Colloquium*. 1998. Seattle, USA: ACM.

[146] Systinet WASP Server for Java, http://www.systinet.com/

[147] IONA Artix, http://www.iona.com/

[148] FusionWare Integration Server, http://www.fusionware.net

[149] Sun Java WSDP, http://java.sun.com/webservices/

[150] NetBeans, http://www.netbeans.org/

[151] Eclipse, http://www.eclipse.org/

[152] A comparison between Relational and Object Oriented Databases for object oriented application development, http://www.tietovayla.fi/borland/cplus/revquide/oo1_rel.htm

[153] The Object-Oriented Database System Manifesto, http://www-2.cs.cmu.edu/People/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html

[154] Pereira, J.L., *Tecnologia de Bases de Dados*. Tecnologias de Informação. 1997: FCA.

[155] Java Media Framework, http://java.sun.com/products/java-media/jmf/

[156] Java Shared Data Toolkit, http://java.sun.com/products/java-media/jsdt/

[157] Java Data Objects (JDO), http://java.sun.com/products/jdo/

[158] FastObjects Trial Edition, http://www.versant.net/eu_en/Downloads/FastObjectsTrialEdition

[159] RDF, http://www.w3.org/RDF/

[160] Dublin Core Metadata Initiative, http://dublincore.org/

[161] MPEG-7, http://www.mpeg.org/MPEG/starting-points.html#mpeg7

[162] ODMG OQL User Manual, http://www.odmg.org/oqlg.zip

[163] OQL - Object Query Language, http://www.db.ucsd.edu/People/michalis/notes/O2/OQLTutorial.htm

[164] Austerberry, D., *The Technology of Video & Audio Streaming*. 2002: Focal Press. 331.

[165] Ohanian, T.A., *Digital Nonlinear Editing*. Second Edition ed. 1998: Focal Press.

[166] Oliveira, P., B. Fonseca, and E. Carrapatoso. *Distributed Architecture for an MPEG2 TV Studio*. in *XV Simpósio Brasileiro de Telecomunicações*. 1997. Recife, Brasil: Sociedade Brasileira de Telecomunicações.

[167] Brightwell, P. *A Distributed Television Production Environment using MPEG-2 Compression and IT Infrastructure*. in *European Workshop on Distributed Imaging*. 1999. Londres, Reino Unido: IEE.

[168] Linder, H., H.D. Clausen, and B. Collini-Nocker, *Satellite Internet Services Using DVB/MPEG-2 and Multicast Web Caching*, in *IEEE Communications Magazine*. 2000. p. 156-161,

[169] Oliveira, P., B. Fonseca, and E. Carrapatoso. *An MPEG-2 Distributed Studio Architecture Based on ATM*. in *MELECOM'98*. 1998. Tel Aviv, Israel: IEEE.

[170] Fonseca, B., P. Oliveira, and E. Carrapatoso. *Non Linear Editing in an MPEG2 Studio*. in *XV Simpósio Brasileiro de Telecomunicações*. 1997. Recife, Brasil: Sociedade Brasileira de Telecomunicações.

[171] NIST Internet Time Service, http://www.boulder.nist.gov/timefreq/service/its.htm

Índice Remissivo