Universidade Técnica de Lisboa
Instituto Superior Técnico

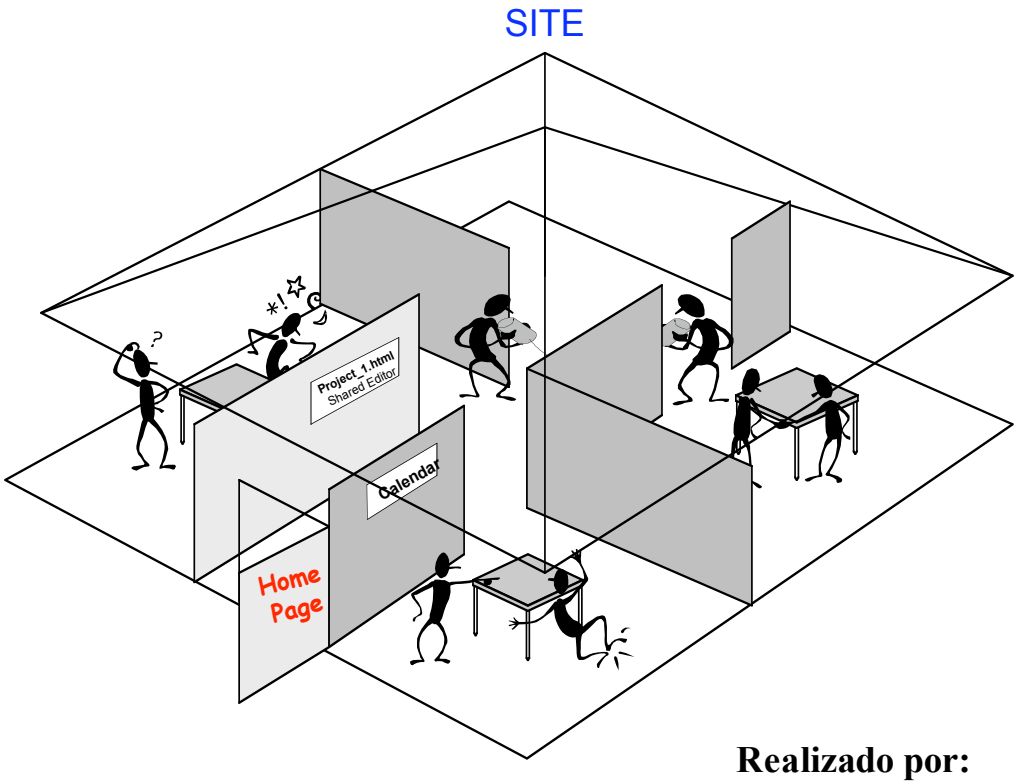
–

Licenciatura em Engenharia Informática e de Computadores

Trabalho Final de Curso ano lectivo 1997/1998

Monitorização de Grupo Utilizando Tecnologias WWW

Relatório Final



Realizado por:

António Pedro Oliveira – 38383

Pedro Ferraz – 38516

Orientado por:

Prof. Pedro Antunes

Outubro 1998

Agradecimentos

Agradecemos a todos quanto nos ajudaram na realização deste projecto nomeadamente ao nosso orientador Professor Pedro Antunes, aos nossos colegas de laboratório: José "Ardiloso" Cardoso, Rui "Ó Zé és sempre a mesma coisa" Pinto, Paulo "Virago" Clemente e João "Trata disso " Pagaime, aos nosso progenitores que ainda nos sustentam, ás nossas namoradas que nos aturaram em momentos difíceis, para todos eles um grande BEM HAJA!

INDEX

1 - Introdução

Hoje em dia, todos estamos cientes do papel da comunicação na colaboração entre grupos de trabalho tendo sempre em vista um objectivo comum; a resolução de problemas e o consequente alcance de metas.

Desde sempre, os povos procuraram estabelecer vias de comunicação que levassem à sua expansão e desenvolvimento, tendo os que as renunciaram ou desprezaram, acabado por desaparecer, ou simplesmente não evoluir.

Ainda hoje, as rotas comerciais de cada país são um factor decisivo para o seu desenvolvimento e progresso.

Tal como a comunicação, a colaboração entre dois povos ou comunidades, sempre se revelou benéfica e vantajosa (pelo menos para uma das partes), indicadora de prosperidade. Como exemplo disto, a passagem de uma economia de subsistência para uma de mercado, foi um marco na história humana.

O evento da explosão dos computadores, baseia-se em grande parte devido às vantagens introduzidas pelo mesmos, nos campos da comunicação e colaboração. O exemplo mais categórico, é sem dúvida a Internet, que tem vindo a reforçar cada vez mais o conceito de aldeia global.

Apesar disto, o computador começou por se evidenciar devido ao seu poder computacional e possibilidades daí resultantes, tendo sido desenvolvidas inúmeras aplicações que ajudam o utilizador a realizar o seu trabalho, tal como folhas de cálculo, processadores de texto e de desenho, etc. .

Só mais tarde, o potencial de meio de comunicação foi explorado, e aí apareceram as aplicações de transferência de ficheiros, correio electrónico, etc., que permitiram às pessoas interagir umas com as outras, estando em locais mais ou menos distantes.

A acompanhar esta evolução apareceu a *World Wide Web* (*WWW*) que entrou em grande parte das organizações, negócios e casas. A *Web* conseguiu gerir com sucesso as questões de heterogeneidade ao fornecer um conjunto de funcionalidades sobre um leque alargado de plataformas. Esta aproximação múltipla dentro da *Web* continua a ter os seus desenvolvimentos.

1.1 – WWW como suporte para trabalho cooperativo

Com toda esta evolução foi necessário criar uma nova categoria de aplicações a qual se deu o nome de **Trabalho Cooperativo** ou, aplicações para colaboração em grupos de trabalho. Esta nova categoria foi definida com a introdução da aplicação *Lotus Notes* [9] no ano de 1989. Esta aplicação surgiu de uma outra denominada *Plato Notes* que se destinava a fóruns centralizados, desta forma podemos afirmar que esta categoria é um desvio ás aplicações de fóruns centralizados. Mas, nas áreas onde as aplicações fórum estavam focadas no início, como por exemplo grupos de discussão, os produtos de trabalho cooperativo suportam uma variedade alargada de outras actividades, tal como escalonamento e partilha de documentos. Alguns produtos de trabalho cooperativo apenas adicionam novas funcionalidades a aplicações já realizadas; outras são caixas de ferramentas para a criação de aplicações colaborativas com modelos personalizáveis incluídos[8].

Podemos referir como exemplos: *Metting Works*, *Groupsystems*, *GroupKit* [4], *Team It*, *Tango*, *BSCW* e *CoopWWW* [11]. Na observação que fizemos destas aplicações tivemos em atenção a forma como todas elas introduziam o trabalho cooperativo com suporte computacional.

O trabalho cooperativo surge com a função de assistir os grupos na comunicação, colaboração e coordenação das suas actividades. Deste modo, este tipo de aplicações tornam-se o elo de ligação entre utilizadores e, implicitamente entre computadores, facilitando e muitas das vezes possibilitando, o trabalho em grupo.

O trabalho cooperativo suporta três vertentes de actividades:

- Intercomunicação pessoal
- Geração de ideias e decisões
- Partilha de objectos.

Pode ser classificado pelo local, tempo e função a que se destina. A matriz espaço/tempo diferencia numa primeira aproximação os vários sistemas, dividindo-os pelo local: mesmo local/ remoto e pelo tempo: simultâneo/ tempos diferentes[1].

Como exemplos de sistemas de trabalho cooperativo pode-se encontrar dentro de cada vertente:

- Intercomunicação pessoal: Email, sistemas de mensagens estruturadas, vídeo conferência, etc. .
- Sistemas de suporte a reuniões: Ferramentas de argumentação, salas de reuniões especialmente preparadas, superfícies de desenho partilhado, etc. .
- Partilha de objectos: PCs partilhados, sistemas de janelas partilhadas, editores partilhados, sistemas de co-autoria, etc. .

1.2 – Identificação do Problema

Com já referimos atrás a Internet tem sido um dos meios de comunicação que mais evoluiu neste últimos anos. Apesar dessa evolução ser enorme, uma desvantagem que se encontra associada aos *browsers* é a sua unidireccionalidade. Podemos afirmar que um dos aspectos mais relevantes da *WWW* é a sua **solidão** (Fig.1).

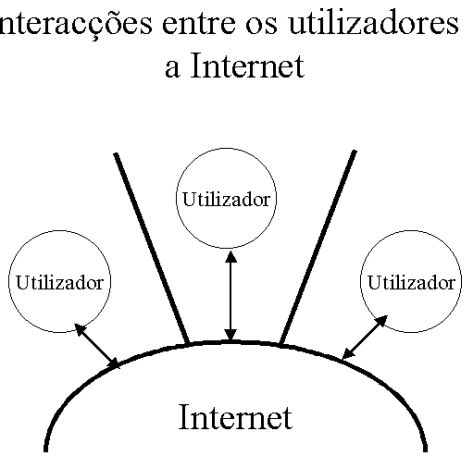


Figura 1 :Interacção entre os utilizadores e a Internet

Ora vejamos, se formos ás compras a um centro comercial podemos encontrar outras pessoas, ver velhos amigos, etc. Apesar do número de utilizadores na *Web* ser bastante elevado, este tipo de situações nunca acontece se decidirmos ir a às compras numa loja virtual.

Outro dos defeitos, directamente relacionado com este anterior, é o caso em que um utilizador desejar alterar uma determinada página, as suas modificações **não** podem ser observadas **simultaneamente** pelos restantes utilizadores que nesse momento a possuem. Têm-se desenvolvido aplicações para minimizar o problema. No entanto, o resultado da rede é que tem que ser o utilizador a explorar essas facilidades. Esta aproximação é aceitável num ambiente homogéneo onde todos os utilizadores estão a explorar o mesmo par cliente servidor.

Uma das grandes lições que nos trouxe a *WWW* foi que a heterogeneidade tem que ser aceite como uma norma. Como consequência temos a necessidade de desenvolver protocolos abertos e de *A.P.I.'s* que suportem aplicações múltiplas. **Protocolos abertos** são uma das formas para obter a flexibilidade essencial para o desenvolvimento de sistemas de trabalho cooperativo e são a base para o desenvolvimento de ferramentas de trabalho.

A *Web* apresenta-se como uma oportunidade tremenda para desenvolver futuras aplicações de trabalho cooperativo com suporte de computadores. Um número significativo de protótipos sistemas já começou de emergir.

Um dos exemplos típicos é um sistema de conferências. Mas este só é baseado na *Web* se grande parte das suas funcionalidades utilizarem *Web browsers* e servidores. No entanto, esta distinção ainda não se encontra tão clara tal como soa. Grande parte das pessoas que estão a desenvolver aplicações de conferência têm pressa em adaptar os seus produtos à Web. Como resultado obtemos um conjunto de

produtos híbridos que utilizam a Web em grande ou pequena extensão. Sendo assim, não ainda existe uma fronteira bem definida entre aplicações de conferência com suporte da Web e sem esse suporte[8].

Os sistemas que funcionam com *Web browsers* e servidores não modificados são de grande interesse, uma vez que eles são a forma mais fácil de aceder à grande massa da *Web* e utilizadores da Intranet. Qualquer aplicação da *Web* que requer do utilizador a instalação de um apêndice ao *browsers* ou mesmo um novo *browser*, sofre uma incapacidade, mesmo que essa aplicação específica possa trazer uma melhoria significativa na interface do utilizador[8,2].

Isto traduz assim uma certa inércia por parte dos utilizadores e com razão, já que não é viável a instalação de novos componentes e apêndices, por cada aplicação desenvolvida na Internet. O próprio conceito de Internet vai mais de encontro a uma **normalização**, do que a uma diversificação…

A *Web* suporta deficientemente actualizações em tempo real. A sincronização na partilha de informação é também dificultada devido à distância que poderá existir entre os diversos intervenientes, introduzindo grandes tempos de latência.

Como conclusão, pudemos afirmar que o número de ferramentas que nos permitem cooperar através da *Web* é reduzido e disperso, sendo este um campo ainda por **explorar** e sobretudo, **normalizar**.

2 - Sistema Idealizado

O nosso projecto tem como principal objectivo o desenvolvimento de uma aplicação de cooperação baseada na *WWW*, a qual fornece uma plataforma de serviços para colaboração em grupos que utilizam a tecnologia *Web* existente, transformando assim, a *Web* de um armazém primário de informação passiva numa ferramenta de **cooperação activa**.

O seu papel é o de monitorizar trabalho cooperativo sobre a plataforma *WWW*.

Desta forma, as páginas tornam-se **salas de convívio** onde se pode entrar, sair, **encontrar** e **interactuar** com outras pessoas sejam elas conhecidas ou não.

Interacções entre os utilizadores e a Internet ideal

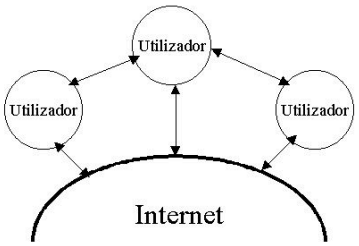


Figura 2 :Interacções ideais entre os utilizadores e a Internet.

2.1 Funcionalidades

Idealmente, o servidor de cooperação implementa diversos **tipos de funcionalidades** tais como:

- **Controlo de interacção**
- **Suporte à cooperação**
- **Controlo de acesso**

Os utilizadores interactuam com o servidor através de um *browser*. Este, por sua vez, auxilia o utilizador nas diversas funções que vão ser disponibilizadas pela aplicação partilhada. Esta interacção será integrada noutras aplicações partilhadas

como por exemplo editores de texto, folha de cálculo, multimedia, editores de imagens, etc.

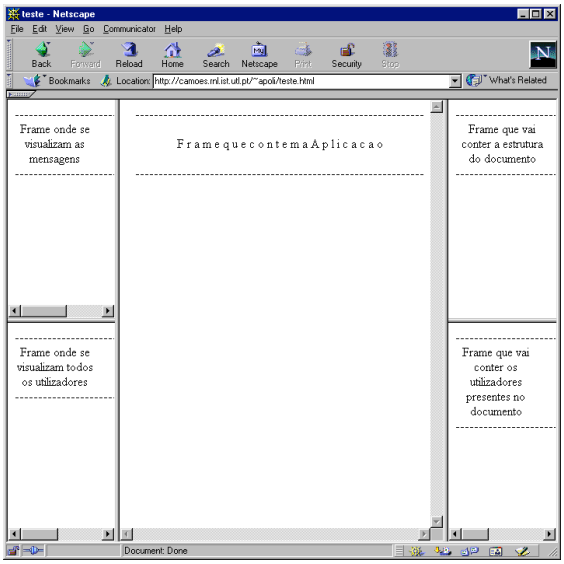


Figura 3 : Ambiente integrado ideal

Assim, o utilizador depara-se com um ambiente **integrado** no *browser*, constituído pela interface de monitorização, e pela janela de *browser*, como hoje a conhecemos (Fig.3).

A interface de monitorização é constituída por uma janela de mensagens onde aparecem comunicações de outros utilizadores ou do servidor, uma janela de utilizadores activos que indica quem está no *site*, outra semelhante mas que apenas indica os que estão no documento e finalmente uma janela de estrutura onde se pode visualizar, caso o nível de operacionalidade o permita, a estrutura do documento.

No esquema que se segue descreve-se a interacção entre as diversas entidades envolvidas na aplicação.

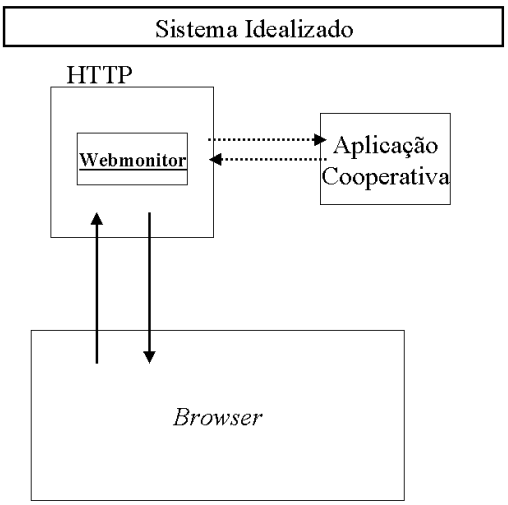


Figura 4 : Interacção entre as entidades

Nesta figura pode-se constatar que no caso ideal o monitor é uma parte integrante do servidor HTTP, pois só assim se poderá obter uma hegemonia entre as duas partes. A nossa aplicação será assim uma extensão ao actual servidor HTTP.

A comunicação entre o servidor do monitor e o servidor da aplicação é feita em um dos três níveis de integração possíveis, consoante o tipo de interacção existente entre o monitor e a aplicação partilhada:

- **Nível zero (0)** :Nível mais baixo, destinado a aplicações que não tenham qualquer tipo de interacção com o monitor. O caso de páginas estáticas enquadra-se neste nível. O servidor do monitor lança o servidor de aplicação caso este exista, e deixa de haver interacção entre ambos.
- **Nível um (1)** : O servidor da aplicação serve-se da API do servidor de cooperação para enviar mensagens para os utilizadores, informando-os acerca de eventos relacionados com o documento visualizado.
- **Nível dois (2)** : Além de mensagens, existe também informação referente à estrutura do documento. O utilizador tem assim informação acerca de como o documento está a ser alterado em termos da sua estrutura.

Como podemos verificar, o sistema idealizado é composto por duas entidades separadas, mas que podem interligar-se entre si. Estas entidades são o nosso par servidor/cliente de cooperação, e uma aplicação partilhada.

O sistema de suporte ao trabalho cooperativo serve assim de **elemento de suporte** às aplicações partilhadas.

O objectivo é ter um ponto comum que dê suporte, e ao mesmo tempo tente minimizar ou mesmo superar as deficiência na Internet nos termos a que diz respeito à **percepção**[1] do meio em redor e **troca de informação** no sentido **utilizador -> *browser*** e também no sentido **utilizador -> utilizador**.(ver Fig.2)

A opção por este três níveis serve para uma melhor estruturação da integração a ser realizada posteriormente para aplicações mais genéricas. Isto é, compatibilizar a nossa aplicação com outras. Desta forma, podemos obter níveis de integração consoante o grau de compatibilidade que desejamos obter. Como exemplos podemos referir a aplicação *Calendar* que possui um nível de integração zero, com nível um temos como exemplo a aplicação *Guestbook* ,em quanto o *SharedEditor* de nível dois.

2.2 – WebMonitor

A arquitectura desta aplicação é cliente servidor. O cliente Webmonitor é a interface que permite aos utilizadores comunicar com o servidor. Esta, por sua vez, consiste num conjunto de janelas integradas no *browser* que informa o utilizador acerca de eventos provenientes de outros utilizadores, assim como de dados relativos ao servidor da aplicação que nesse instante está a ser monitorizada. Também permite obter informação útil acerca dos utilizadores.

[1]Percepção tem o mesmo significado que Awareness em Inglês.

A identificação de um utilizador perante os outros fica sempre dependente do próprio mesmo. Desta forma, estamos a preservar a integridade e anonimato das pessoas (a obtenção do número IP de utilizador pode ser mal utilizado por pessoas menos escrupulosas).

2.3 – Servidor

Esta entidade tem por missão implementar as funcionalidades descritas no início deste capítulo. Nunca esquecendo que é uma parte integrante do servidor HTTP, pois só desta forma se consegue obter valores máximos em termos de transparência, portabilidade, fiabilidade e etc.

O **protocolo** utilizado nas interacções entre os clientes e o servidor é constituído por um conjunto **simples** e **curto** de mensagens. Assim, o desempenho da rede não é comprometido ao fazer a transição do servidor HTTP simples para o estendido.

3 - Requisitos

3.1 – A plataforma WWW

Existem diversas vantagens na utilização da Web como base de suporte para partilha de informação colaborativa[2]. Nomeadamente se o ambiente for caracterizado pela existência de diversas plataformas. Como exemplos temos:

- Os *browsers* estão disponíveis para todos os sistemas operativos mais populares; fornecendo acesso a informação de uma forma independente da plataforma.
- Os *browsers* oferecem uma interface de utilizador único e apresentam a informação de uma forma consistente ao longo destas plataformas.
- As estruturas de dados que não podem ser apresentadas directamente pelo browser podem ser facilmente delegadas a aplicações "de ajuda" externa para o seu processamento.
- *Web browsers* actualmente já fazem parte do ambiente de um número crescente de organizações, não necessitando de instalações adicionais ou manutenção das aplicações para os utilizadores colaborarem utilizando a *Web*.
- Muitas organizações também instalaram os seus próprios servidores de *WWW* como parte de uma presença na Internet ou na Intranet e está familiarizada com a manutenção do servidor e, em muitos casos, uma extensão do servidor através da programação da API do servidor.

Dadas estas características, a extensão da *Web* para providenciar formas mais ricas de suporte para cooperação em grupos de trabalho é em ambos apropriado e desejável. Este tema é o foco deste projecto, o qual tem por objectivo principal transformar a Web a partir de um armazém primário de informação passiva torná-lo numa ferramenta activa de cooperação.

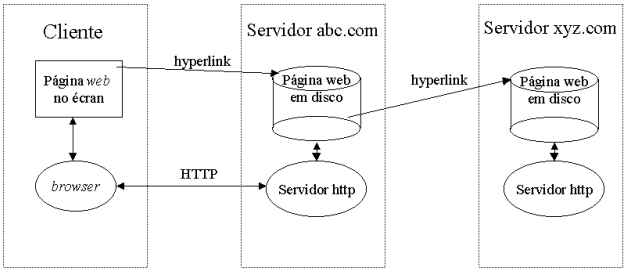


Figura 5 : Exemplo da arquitectura da Web.

3.2 – Interfaces dos Utilizadores

Neste caso, podemos afirmar que o requisito necessário é possuir um *browser.*

Existem **dois tipos** de interface com os utilizadores na nossa aplicação:

- Janela do *browser* com a aplicação cooperativa;
- Janela com o monitor de trabalho cooperativo;

O primeiro é da responsabilidade da aplicação partilhada e o segundo é da responsabilidade do **Webmonitor**

4 - Restrições

4.1 – Tecnologias disponíveis

Para encontrar soluções para o nosso **problema**, tivemos sempre em mente a falta de interactividade entre os utilizadores, desta forma, verificámos a existência de diversas tecnologias, muitas delas bastante recentes. Tivemos em consideração determinados factores **característicos** nessas mesmas tecnologias, como por exemplo **portabilidade, compatibilidade** e **robustez**. No entanto, nem sempre fomos bem sucedidos, de seguida descrevem-se as situações mais relevantes:

4.1.1 - HTTP → Cliente → Servidor

Um dos critérios utilizados no desenho do protocolo a implementar nas ligações entre as diversas entidades é que este deveria ser simples. Como simples nós entendemos que o número de mensagens a circular deveria ser mínimo e se possível sem estado. Se o protocolo for com estado a sua implementação é mais complexa as hipóteses de erro são mais reduzidas.

O **protocolo é pedido – resposta**. Todos os pedidos com origem no cliente e todos os pedidos irão receber alguma resposta do servidor. A resposta pode ser um simples reconhecimento, um resultado de uma questão ou simplesmente uma mensagem de erro.

Para manter esta estrutura simples, o protocolo é implementado sobre o Protocolo de Controlo de Transmissão / Protocolo Internet (TCP/IP). TCP/IP é orientado á ligação e necessita de menos verificações no servidor, diminuindo desta forma a complexidade do servidor.

Se for esperado um elevado número de mensagens a atravessarem a rede, estas provocarão o aumento do tempo de inicialização para este tipo protocolo. Desta forma, dever-se-á utilizar um protocolo menos seguro, *USER DATAGRAM PROTOCOL* (UDP)[5].

4.1.2 – *Common Gateway Interface*

Apesar da inexistência de suporte para colaboração directa, os protocolos e modelos correntes da *Web* escondem muita da complexidade no desdobramento de aplicações num ambiente distribuído e heterogéneo.

O método mais comum para a sua realização é estender um modelo de servidor *Web* através da *COMMON GATEWAY INTERFACE* (CGI) com uma nova funcionalidade da aplicação ou um código "cola" a uma aplicação já existente, apresentando uma "interface" à aplicação como uma série de páginas *HTML* as quais podem ser apresentadas por um *Web browser*. Com este método, os programadores podem adquirir a vantagem da base já existente dos browsers como programas

clientes para as suas aplicações, e outras vantagens listadas em baixo, mas também devem aceitar as desvantagens da arquitectura básica cliente – servidor da *Web* como também dos seus protocolos e as limitações correntes dos próprios *browsers*[2].

Estes constrangimentos inibem o desenvolvimento de certo tipo de aplicações, particularmente aquelas que necessitam de:

- **Media contínuo**: *HTTP* não suporta directamente a especificação de uma taxa garantida de transmissão entre o servidor e o cliente. Em geral, transferência de dados resulta na transmissão de conjuntos de tramas os quais não são comportáveis para media contínuo (tempo - real) como som e imagem.
- **Replicação de informação**: A arquitectura não fornece suporte directo para a replicação de informação, a qual pode ser requerida para trabalhar sem ligação, feedback rápido ou segurança.
- **Comunicação utilizador – utilizador**: não existe nenhum suporte para comunicação servidor – servidor, (iniciação do servidor) servidor – cliente ou cliente – cliente, problemático para aplicações onde o servidor necessita de ter um papel activo (por exemplo notificação dos utilizadores de alterações para informação ou para manter a consistência na informação sobre múltiplos servidores).
- **Interfaces de utilizadores ricas**: Mesmo o *HTML* suportar funcionalidades tais como simples caixas de preenchimento de formulários, não é um conjunto de ferramentas para interface com o utilizador e a interacção de estilos que podem ser suportados são muito limitados.

4.1.3 – *Java*

O instrumento utilizado para o desenvolvimento do nosso projecto recaiu sobre o ***Java***.

4.1.3.1 – A linguagem

Java é na verdade valioso para ambientes de redes distribuídas como a *Web*. No entanto, o *Java* vai além deste domínio ao fornecer uma linguagem de programação poderosa e generalista aplicável na construção das mais variadas aplicações que não necessitam das funcionalidades da rede ou necessitam delas por diferentes razões. A capacidade do *Java* executar código em máquinas remotas de uma forma segura é um requisito crítico para muitas organizações[3].

Outros grupos utilizam o *Java* como uma linguagem de programação de âmbito geral para projectos onde a dependência da máquina é menos importante. A

facilidade de programação e a segurança do *Java* ajudam na produção rápida de código para detecção de erros. Alguns dos erros mais comuns de programação não acontecem em virtude da existência de funcionalidades como *Garbage Collection* e referência seguras para tipos.

Aplicações modernas baseadas na rede e nas aplicações gráficas baseadas das interfaces do utilizador que devem executar múltiplas tarefas de forma simultânea é satisfeito pelo suporte de *Java* para *Multithreading*, enquanto os mecanismos de tratamento de excepções facilita a tarefa de tratamento de condições de erro.

Enquanto as suas ferramentas de construção são poderosas, a linguagem *Java* é por si mesmo simples na qual os programadores se podem tornar rapidamente bastante eficientes.

4.1.3.2 – O Ambiente de desenvolvimento

O *Java* foi desenhado para ter o máximo de portabilidade e especificamente desenhado para ter o mínimo possível de dependências entre implementações. Ao definir todo o que for possível sobre a linguagem e o seu ambiente de execução permite aos utilizadores executarem o código compilado em qualquer lado e partilhar scódigo com qualquer pessoa que contenha um ambiente *Java*.

Esta linguagem foi universalmente aceite pelos utilizadores da Internet e da *WWW*. Desta forma, os utilizadores beneficiam do acesso a uma plataforma segura e independente da aplicação que pode vir de qualquer lado da Internet. Os programadores ao criarem aplicações em *Java* o código é gerado apenas uma vez, sem a necessidade de transportarem as suas aplicações para cada plataforma (sistema operativo e máquina)[3].

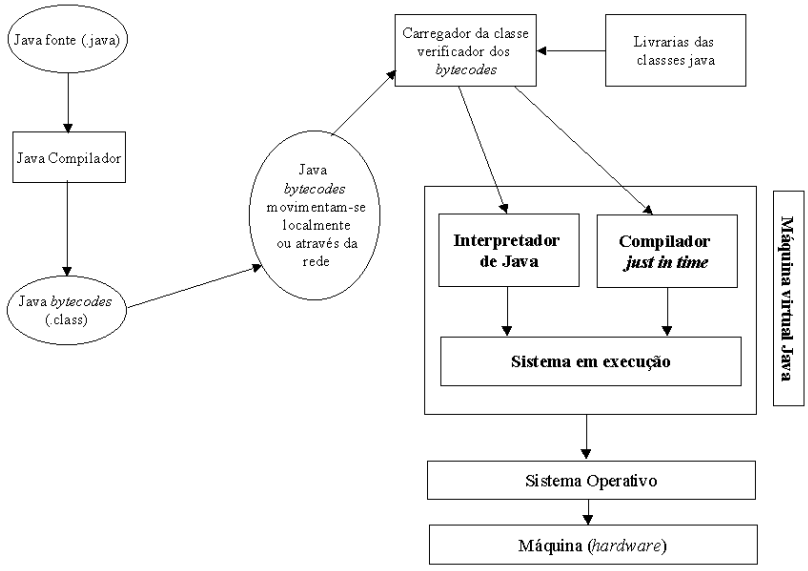


Figura 6 : Exemplo do comportamento do interpretador/compilador[6] de Java

4.1.4 - *Applets*

Uma *applet* é um programa *Java* que se integra numa página *WWW*. Uma vez que *Java* é uma linguagem segura, as *Applets Java* podem ser carregadas a partir da rede e executadas sem muitas preocupações.

Existem dois problemas principais no desenvolvimento de versões *Java* nos clientes.

Em primeiro lugar, para desenvolver um cliente de percepção (*Awarness)* como uma *Applet Java* é necessário que ela esteja persistentes entre duas localizações. Com isto nós queremos dizer que quando um *Web browser* se move da localização onde a *Applet* se encontra localizada, a *Applet* deve continuar a sua execução até ser terminada pelo o utilizador ou o *browser* é fechado.

O segundo problema consiste no seguinte: na altura da escrita, uma *Applet Java* apenas pode abrir um canal de comunicação com o servidor no qual se encontra o seu código fonte. Qualquer outra tentativa de abrir um canal de comunicação para outro local viola o Modelo de Segurança.

Até que a comunidade de *Java* consiga ultrapassar estes problemas torna-se difícil o desenvolvimento de clientes *Java* "limpos" para aplicações de *Awarness*. No entanto, esta foi a nossa escolha, mais tarde poderá entender o porquê [5].

4.1.5 - *Hypertext Markup Language*

Hypertext Markup Language (HTML) é a linguagem utilizada para codificar documentos na *WWW* . É a apresentação de um documento e uma linguagem de especificação de *hyperlink* que define a sintaxe e a colocação de direcções especiais embebidas que não são apresentadas por um *Web browser.*

Também lhe diz como apresentar o conteúdo do documento, incluindo texto, imagens, e outros suporte para aplicações multimedia. A linguagem também afirma como tornar o nosso documento interactivo através de ligações *hyperlink* especiais, as quais ligam o nosso documento com outros documentos no nosso sistema local, a *WWW*, e outros recursos da Internet como o *FTP* e *Gopher*.

A sintaxe e semântica base do *HTML* são definidos no modelo (*standard*) do *HTML*. O modelo do *HTML* e outros modelos relacionados com a *web* são desenvolvidos sobre a autoridade do *World Wide Web Consortium* (W3C) [7, 10].

4.1.2 – *Web browsers*

Como já foi referido no início do capítulo anterior (capíulo 3) a principal interface com os utilizadores é o ***browser***.

No entanto, este terá de suportar o *Java* versão 1.2 para que o nossa aplicação seja executada. Como exemplos de utilização podemos referir o browser da Netscape a partir da versão 4.0 ou a versão 4.0 do Internet Explorer..

Contudo, foi verificado que apenas no browser da Netscape todas as funcionalidades se encontravam presentes. Em relação aos outros browsers não nos foi permitido verificar com mais pormenor a correcta execução do nosso monitor.

5 - Trabalho Realizado

A nossa primeira abordagem ao problema consistiu na definição do conceito de trabalho cooperativo : o quê?, para quê?(ver capítulos anteriores). De seguida, fomos procurar aplicações existentes relacionadas com este conceito (ver exemplos na página 4).

A conclusão que retirámos foi a existência de variadas aplicações, no entanto, olham para a Web como um meio de transmissão e não como uma ferramenta de construção como seria o desejável. O que significa esta afirmação é que **não** existe nenhuma aplicação que consiga fazer estes **dois tipos** de eventos ao mesmo tempo: permitir a dinamização das páginas da Web (CoppWWW) e a discussão de ideias entre os participantes (groupkit).

A seguir, decidimos iniciar a construção da nossa aplicação, tendo que enfrentar alguns obstáculos na implementação do sistema, que a seguir se descrevem.

5.1 - Servidor Coordenação

A existência deste servidor tem como objectivo o controlo de toda a informação relativa á monitorização de grupo. Tem como função principal, dependendo do nível de interacção da aplicação que está a ser monitorizada, enviar mensagens para todos os utilizadores que estão num determinado documento ou actualizar a janela que contém informação acerca da estrutura do documento que nesse momento está a ser utilizado.

A opção por este servidor deve-se a melhor estruturação da nossa aplicação. Também se deve ao facto de tornar esta o mais genérica possível, isto é, poder suportar outras aplicações que queiram utilizar a nossa **principal função**, a de **monitorização**.

Desta forma, o nosso servidor serve de API para outras aplicações e estas apenas têm de se adaptar ao nosso servidor para usufruírem das suas potencialidades (mensagens, estrutura do documento). Caso contrário, pode-se utilizar apenas o nível zero da aplicação *Webmonitor*.

Assim, fundamentalmente, o seu papel é o de **coordenador** e **centralizador**, já que é nele que grande parte da informação respeitante aos documentos abertos e utilizadores é guardada.

5.2 - Monitor

Surgiram, ao longo do desenvolvimento do projecto, obstáculos de maior ou menor dimensão. Um dos maiores foi a actualização de páginas no *browser* por parte do servidor. Tentámos várias soluções.

Aquela que se aproxima mais da solução ideal foi a por nós implementada, embora com todas as suas restrições. Na janela de mensagens, sempre que existe uma alteração na página (como exemplo temos a introdução de um novo parágrafo na

aplicação *SharedEditor*, ver capítulo 8) é introduzida uma mensagem do sistema a avisar todos os utilizadores que surgiu uma alteração nesta. Sendo a actualização da mesma feita de forma automática.

As outras alternativas ensaiadas por nós não eram as mais adequadas para a nossa aplicação por diversos motivos.

Uma das soluções seria a utilização de *Common Gateway Interface* (CGI) a dificuldade que nos surgiu foi a impossibilidade de fazer o envio para todos os outros utilizadores que nesse instante estavam a observar essa página.

Outra das soluções seria a utilização de *Javascript* que iria permitir associar um temporizador à respectiva página e desta forma, o browser iria ficar encarregado de contactar o servidor *HTTP* para proceder ao carregamento da página.

Os motivos que nos levaram à não utilização desta última solução foram essencialmente dois: o primeiro, é de certa forma óbvio, será o tempo que iria ser atribuído ao temporizador, sendo o segundo o atraso que iria provocar na aplicação.

Para se determinar um tempo para o temporizador teria que se proceder à recolha de dados estatísticos para se obter um valor aceitável. No entanto, esse valor poderia ser óptimo para utilizadores próximos mas completamente descabido para utilizadores situados na outra parte do globo. Este utilizador, apenas podia observar o seu browser a carregar páginas e não podia utilizar a aplicação.

Este problema é semelhante à utilização de um temporizador directamente no *HTML* da página.

A solução ideal poderia passar pela utilização da API do *browser* mas como ainda não existe um modelo de *browser* uniformizado, teríamos de adequar todos os *browsers* existentes á nossa aplicação (ora isso é quase inviável senão impossível pois a cada surgimento de um novo *browser* implicaria a alteração da sua API para a nossa aplicação). Esta alternativa iria tornar a nossa aplicação dependente do *browser*. O nosso objectivo é realizar uma aplicação que utilize as vantagens do *browser* e não orientada por este.

A opção por nós implementada consiste na utilização de uma *applet java*. Esta inicia a sua actividade contactando o servidor de cooperação. De seguida é lançado, no cliente, o monitor que vai permitir o suporte ao trabalho cooperativo entre os diversos utilizadores.

O motivo que nos levou á definição de uma *applet* foi a possibilidade de comunicação entre o cliente e o servidor e vice-versa, sendo esta última a mais importante já que qualquer uma das outras soluções não permite que sejam executadas acções do servidor no cliente.

Outro motivo para a escolha da *applet* foi a necessidade de uma interface amigável com o utilizador, apesar de alguma rigidez característica da classe *awt* esta mostrou-se ser a melhor solução.

De referir, como nota, que o aparecimento do *swing* vem melhorar substancialmente a qualidade do *Java* no que respeita á interface gráfica. Com alguma

tristeza nossa, o aparecimento desta classe surgiu numa fase terminal do projecto não nos sendo possível explorar as suas vantagens.

5.3 - Interacção Servidor - Cliente e Servidor HTTP

A forma como a comunicação é feita entre os clientes e os servidores é realizada de duas maneiras .

Entre o cliente da aplicação e o seu servidor a ligação pode ser feita através do servidor HTTP (no caso, da aplicação ser o calendário) ou então contacta directamente o servidor da aplicação (no caso, da aplicação ser do tipo *SharedEditor*).

Na interacção do cliente do WebMonitor com o respectivo servidor a comunicação é directa. Estas opções foram tomadas para obter o máximo de performance no protocolo implementado (pergunta – resposta).

Estas comunicações são utilizadas para troca de dados entre as entidades respectivas. Algumas mensagens são respeitantes a informações acerca do que se está a passar nos outros clientes para que desta forma se possa concretizar a troca de dados entre os mesmos.

Como já foi referido atrás existem dois tipos de *sockets* :orientados á ligação e não orientados á ligação. Os primeiros são utilizados nas ligações entre o cliente *Webmonitor* e o respectivo servidor.

A implementação dos níveis de interacção foi realizada através de um protocolo pergunta – resposta e o meio que utilizam nas suas comunicações são os *sockets* orientados á ligação.

6 - Arquitectura + Código + API

6.1 – Arquitectura

Na figura que se segue podemos observar como os diversos componentes se relacionam entre si.

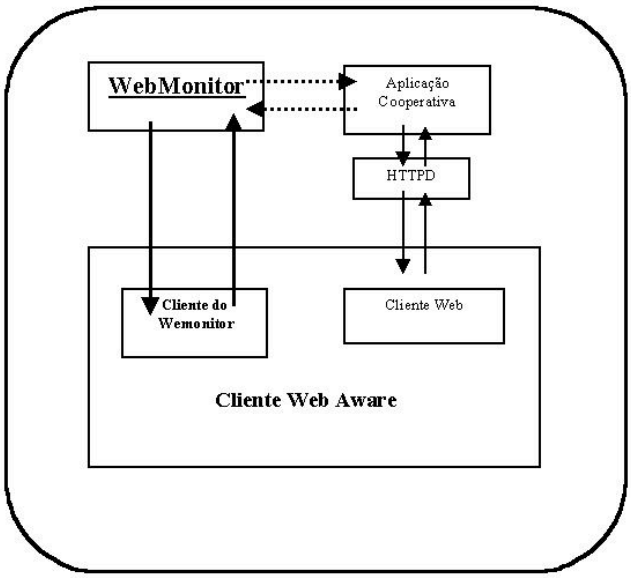
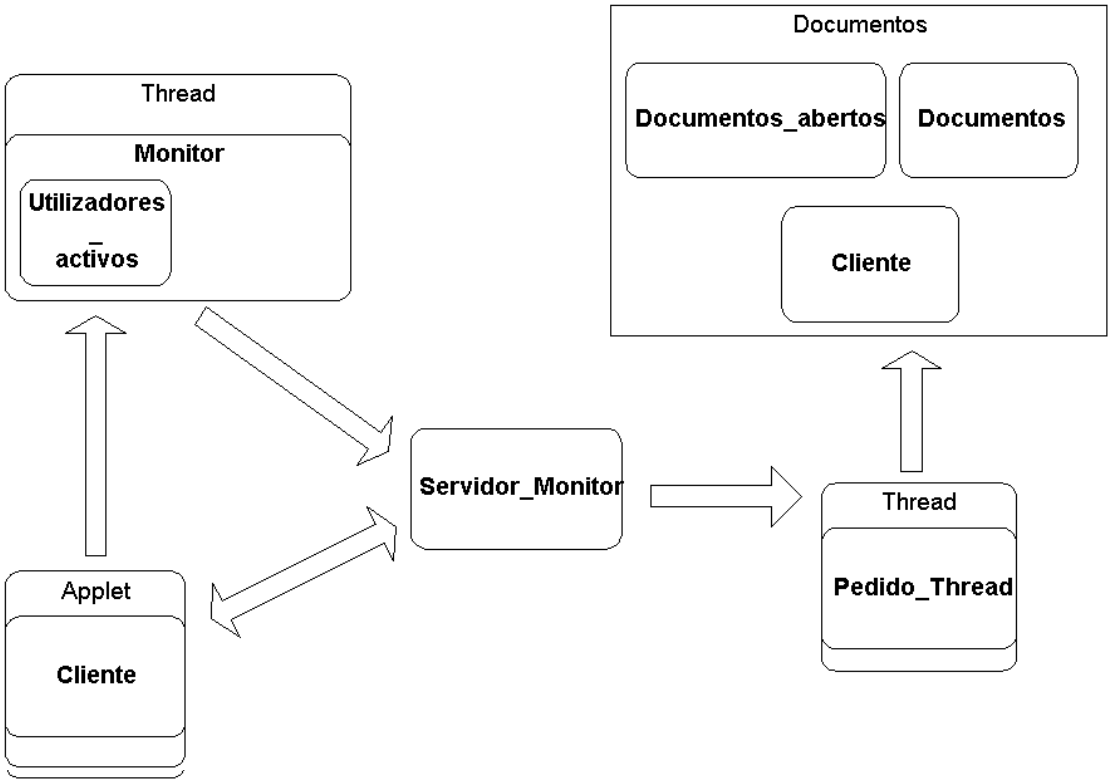


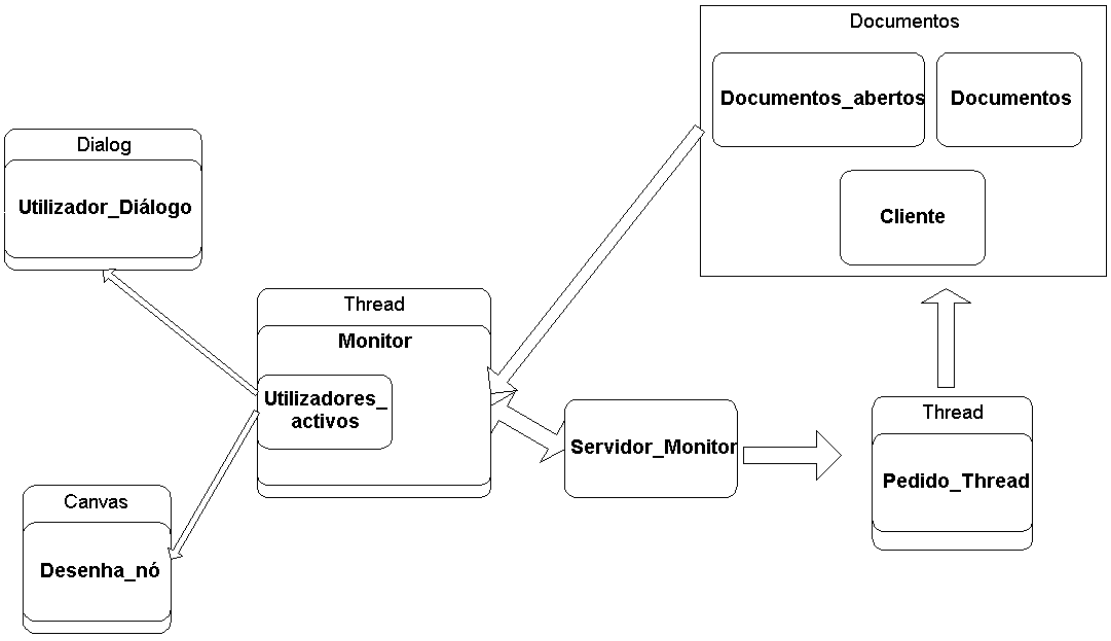
Figura 7 : Arquitectura da nossa aplicação de Percepção

6.2– Hierarquia das classes



Dependências existentes na inicialização do programa (registo do cliente):

Dependências existentes após a inicialização do programa:



6.3 – Estruturação do código

Esta classe implementa a *applet* da página inicial do registo de cliente:

Public Class **Cliente** extends **Applet**
Métodos :
Private Boolean **contact_server**() # Verifica estado do servidor
Private String **getDocList**() # Permite a obtenção da lista de documentos abertos
Public void **init**() # Inicialização da applet
Public void **actionPerform**(ActionEvent) # Apanha todos os eventos gerados pelos componentes da applet
Public void **stop**() # Utilizado pela applet para a sua terminação

Classe que implementa a interface do utilizador:

Contém uma subclasse que gere os utilizadores activos e simultaneamente os que estão no documento.

Public Class **Monitor** extends Thread

Construtores :

Monitor (titulo, cliID, DocNome, ContextoApplet, URL_Inicial) # Definição da interface

Métodos :

Private String **getDocList**() # Permite a obtenção da lista de documentos abertos

Public void **run**() # Espera por eventos vindos do servidor

Public void **actionPerformed**(ActionEvent) # Apanha todos os eventos gerados pelos componentes do monitor

Public void **write_message** (area_texto, mensagem) # métodos utilizados para

Public void **window_closed** (window_event) apanhar eventos na janela

Public void **window_closing** (window_event)

Public void **window_opening** (window_event)

Public void **window_iconified** (window_event)

Public void **window_deiconified** (window_event)

Public void **window_Activated** (window_event)

Public void **window_Deactivated** (window_event)

Public void **Draw_Structure** (vector, url_imagem) # desenha a estrutura do documento

Private Class **Utilizadores_activos**

Construtores :

Utilizadores _activos()

Métodos :

protected void **Adiciona_utilizador_activo** (Nome, Painel)
protected void **Remove_utilizador_activo** (Nome, Painel)
protected void **Adiciona_utilizador_doc** (Nome, Painel)
protected void **Remove_utilizador_doc** (Nome, Painel)
protected void **Mouse_Click** (Mouse_event)
protected void **Mouse_Entered** (Mouse_event)
protected void **Mouse_Exited** (Mouse_event)
protected void **Mouse_Press** (Mouse_event)
protected void **Mouse_Released** (Mouse_event)

Classe necessária para desenhar um elemento da estrutura na janela respectiva.

Public Class **Desenha_nó** extends Canvas
Construtor : **Desenha_nó**(Norte, Sul, Este, Oeste, Altura, Largura, URL_imagem) **Métodos** : Public void **paint** (graphics) Public void **Update** (graphics)

Classe utilizada para pequenas interacções com o utilizador.

Public Class **Utilizador_Dialogo** extends Dialog
Construtor : Utilizador_dialogo(janela_pai, título, modulo, tipo_dialogo) **Métodos** : Protected void **actualiza_utilizador_destino**(cliID) # Envia mensagem para o utilizador correcto Protected String **actualiza_lista_doc_abertos**(documento) Public void **actualiza_titulo_janela**(titulo_janela) Protected void **actualiza_tipo_janela**(tipo_janela) Public void **actionPerformed**(ActionEvent) # Apanha todos os eventos gerados pela janela de diálogo

Classe que implementa o Servidor que fica à espera de pedidos.

Public Class **Servidor_monitor**
Construtor : **Servidor_monitor**() # Cria o socket principal para atender os diversos pedidos **Métodos** : Public void **run**() # Fica em ciclo infinito e lança uma thread por cada pedido Public void **clean_up**() # Fecha o socket Public static void **main**() # Lança-se a si próprio

Classe que trata dos pedidos que chegam ao Servidor.

Public Class **Pedido_Thread** extends Thread
Construtor : **Pedido_thread**(socket) # Inicializa o socket para comunicação com o cliente **Métodos** : Public void **run**() # Trata pedidos das aplicações ou dos utilizadores

6.3.1 - Pacote Documentos

Pacote construído para definir a estrutura que contém os métodos relacionados com os documentos.

Public Class **Documentos_abertos**

Construtor :

Documentos_abertos() # Inicializa o vector de documentos abertos

Métodos :

Protected int **documento_novo_aberto** (nome_documento, tipo_estrutura, tipo_aplicacao)

Protected String **documento_aplicaçao** (nome_documento) # Dado um documento devolve o tipo de aplicação que lhe está associado.

Protected int **numero_documentos_abertos**()

Protected int **sincronizar_utilizador**(nome_documento, cliID, socket) #Sincronização do ambiente de trabalho do utilizador

Protected boolean **verificar_documento_aberto**(nome_documento)

Protected void **adiciona_cliente_documento**(nome_documento, cliID, socket_cliente)

Protected void **remove_cliente_documento**(nome_documento, cliID)

Protected void **verifica_cliente_em_documento**(nome_documento, cliID)

Private document **busca_documento**(nome_documento)

Protected void **escreve_mensagem_client** (mensagem, cliID) # Escreve uma mensagem para um cliente

Protected void **Messagem_broadcast**(nome_documento, cliID) # Envio de uma mensagem para todos os utilizadores num documento

Protected void **Messagem_brodacast**(cliID) # Envio de uma mensagem para todos os utilizadores no *site*.

Private void **escreve_documento_html**(nome_documento) #Escreve o documento em html

Protected String **busca_documentos_abertos**() # Devolve uma *String* que é a lista de documentos abertos.

Protected void **envia_posicao_paragrafos**(nome_documento) #Envia a estrutura de parágrafos para os clientes num documento.

Protected Vector **busca_posicao_paragrafos**(nome_documento) #Método auxiliar que vai buscar o vector de parágrafos de um documento.

Protected void **Insere_paragrafo**(nome_documento, posicao_parágrafo, cliID) #Insere um novo parágrafo.

Protected void **Apaga_paragrafo**(nome_documento, posicao_parágrafo).

Public Class **Documento**
Construtor : Public **documento**(nome_documento, tipo_string, aplicacao) # Public **documento**(documento) **Métodos** : Public String **obtem_nome_documeno**() Public String **obtem_aplicacao_doucmento**() Public int **adiciona_novo_cliente** (cliID, socket_cliente) Public void **remover_cliente**(cliID) Public void **Messagem!!!!_broadcast**(mensagem) # Faz o *broadcast* de uma mensagem para todos os utilizadores no *site*. Public void **envio_mensagem_cliente**(mensagem, cliID,) #Sincronização do ambiente de trabalho do utilizador Public void **verifica_cliente**(cliID) Public void **resincronizar_cliente**(cliID, socket_cliente) Private void **escreve_documento_html**(nome_documento) #Escreve o documento em html

Public Class **Cliente**
Construtor : Cliente(cliID, socket_cliente) Cliente(cliente) **Métodos** : Public int **escrever_mensagem**(mensagem) Public void **resincronizar**(socket_cliente)

6.4 - *API*

Definição dos formatos das mensagens utilizados no protocolo cliente (Webmonitor ou Servidor de aplicação partilhada) → Servidor *Webmonitor* :

STATUS# - Verifica o estado do Servidor.

REGISTER#DocName#CliID#AppName# - Regista um novo cliente num documento gerido por uma aplicação.

LISTENING#DocName#CliID# - Resincroniza um cliente no documento.

SENDMESSAGE#CliID#DestinCli#Msg# - Envia uma mensagem para outro utilizador.

BRDCSTMSGDOC#DocName#CliID#Msg# - Envio de uma mensagem para todos os utilizadores que se encontram num mesmo documento.

RELOAD#DocName# - Dá ordem ao Servidor para ordenar aos cliente que releiam o documento.

BRDCSTMESSAGE#CliID#Msg# - Envio de uma mensagem para todos os utilizadores.

RMVACTVUSER#DocName#CliID# - Remove um utilizador do Servidor *Webmonitor*.

RMVDOCVUSER#DocName#CliID# - Remove um utilizador de um documento.

STRUCT_LIST# DocName#CliID# - Devolve a estrutura de parágrafos.

INSERT_PARAGRAPH #DocName#CliID#Pos# - Insere um novo parágrafo.

DELETE_PARAGRAPH #DocName#CliID#Pos# - Apaga um novo parágrafo.

INFOUSRDOC# DocName#CliID# - Devolve a informação sobre um utilizador.

OPENDOCLIST# - Devolve a lista de documentos abertos.

Definição dos formatos das mensagens utilizados no protocolo Servidor *Webmonitor* → Cliente:

RCVMESSAGE#Sender#Msg# - Recebe uma mensagem do Servidor.

ADDACTVUSER#CliID# - Adiciona um utilizador activo.

ADDDOCUSER#CliID# - Adiciona um utilizador ao documento.

RMVACTVUSER#CliID# - Remove um utilizador activo .

RMVDOCUSER#CliID# - Remove um utilizador do documento.

RELOAD# - Dá ordem para reler o documento.

STRUCT_LIST#StructVector#imageURL# - Informa sobre a estrutura do documento.

7 – Conclusões

O nosso ***Webmonitor*** é uma aplicação para monitorizar o trabalho cooperativo baseando-se em tecnologias *WWW*. Este trabalho mostra que a *Web* pode ser transformada num armazém de informação dinâmica e ser uma ferramenta activa para cooperação, sem comprometer os benefícios da plataforma da *Web* como ferramenta para a partilha de informação.

Podemos afirmar, que a nível das aplicações para trabalho cooperativo na *WWW* existe ainda um novo mundo a descobrir. Como podemos constatar os *browsers* ainda são demasiados estáticos, apesar das tentativas que existem para os tornar mais interactivos, ainda não chegámos a um "bom porto".

Um dos motivos da falta de interactividade dos browsers reside no protocolo existente entre estes e o servidor HTTP. Este servidor tem funções muito limitadas. Ele resume-se a uma aplicação que responde a pedidos de páginas do browser, verifica onde elas se encontram e devolve esse pedido ao browsers, todas as restantes funcionalidades são realizadas pela aplicação cliente.

Ora uma das vantagens da utilização da nossa aplicação centra-se no facto dos utilizadores apenas necessitarem do mínimo de infra-estruturas técnicas. É necessário apenas um *browser*. Desta forma, a aplicação torna-se disponível para um número mais elevado de utilizadores.

No entanto, a aplicação *BSCW* [11] também utiliza os mesmos recurso, mas uma falha importante centra-se no facto da não existência de comunicação entre os utilizadores. Apesar de estes conseguirem partilhar o mesmo documento não podem discutir ideias e defender os seus pontos de vista. Este ponto é muito importante no desenvolvimento de um trabalho cooperativo.

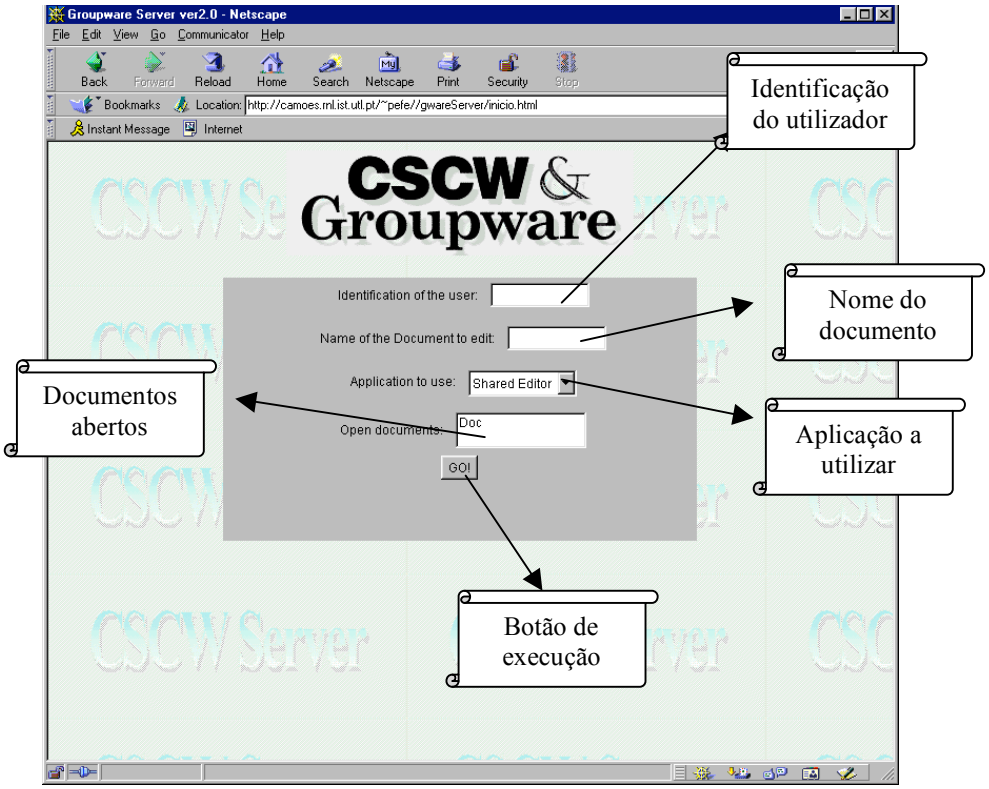
Comparando o *Webmonitor* com outras aplicações como por exemplo *Groupkit* ou o *TeamIt* temos que estas eram aplicações fechadas, isto é, não possibilitavam a interacção com outros programas.

Assim a nossa aplicação vem ocupar um espaço primitivo, pouco trabalhado e aberto a novas ideias.

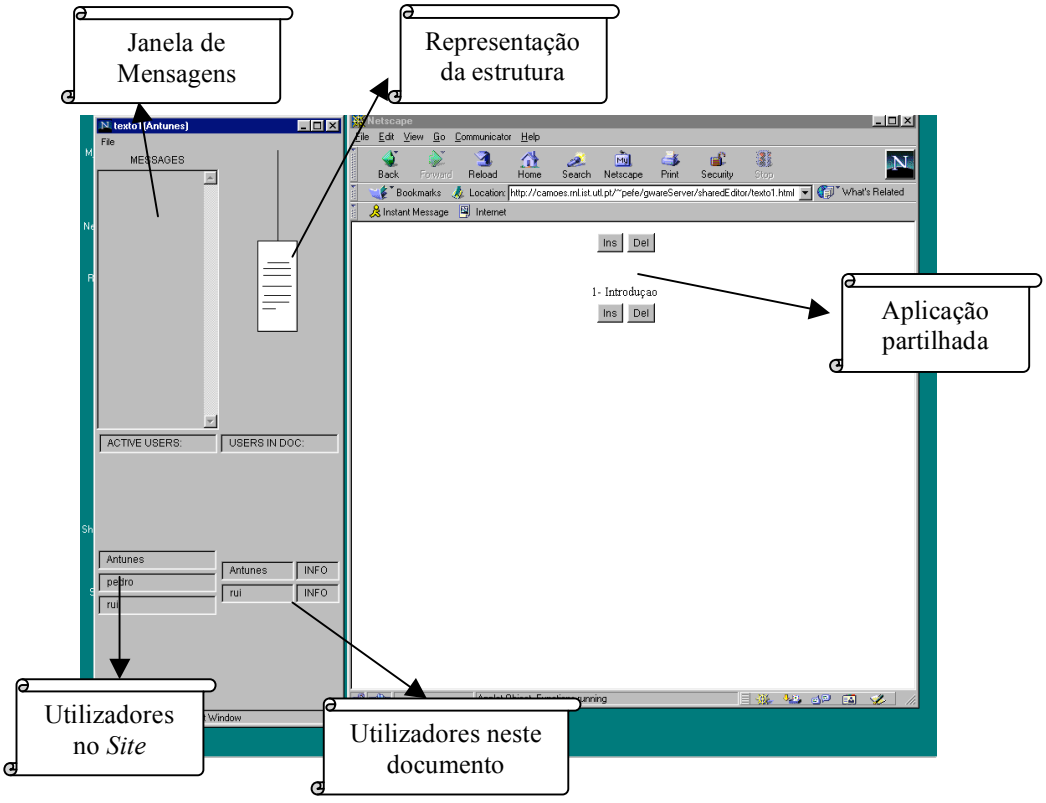
8 – Demonstração da Aplicação

Sequência de figuras exemplificativas da utilização da aplicação.

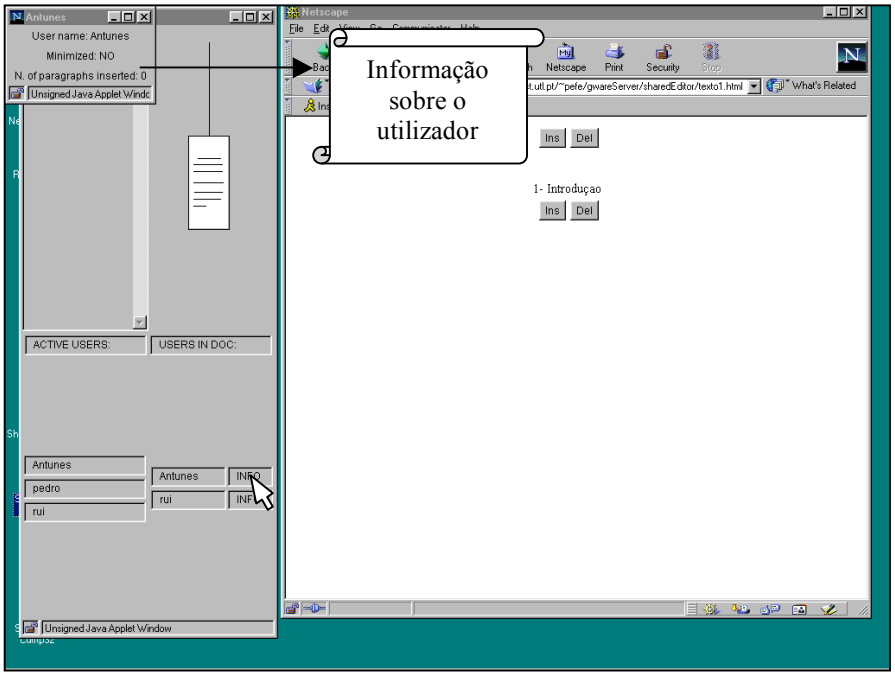
Página inicial da nossa aplicação:



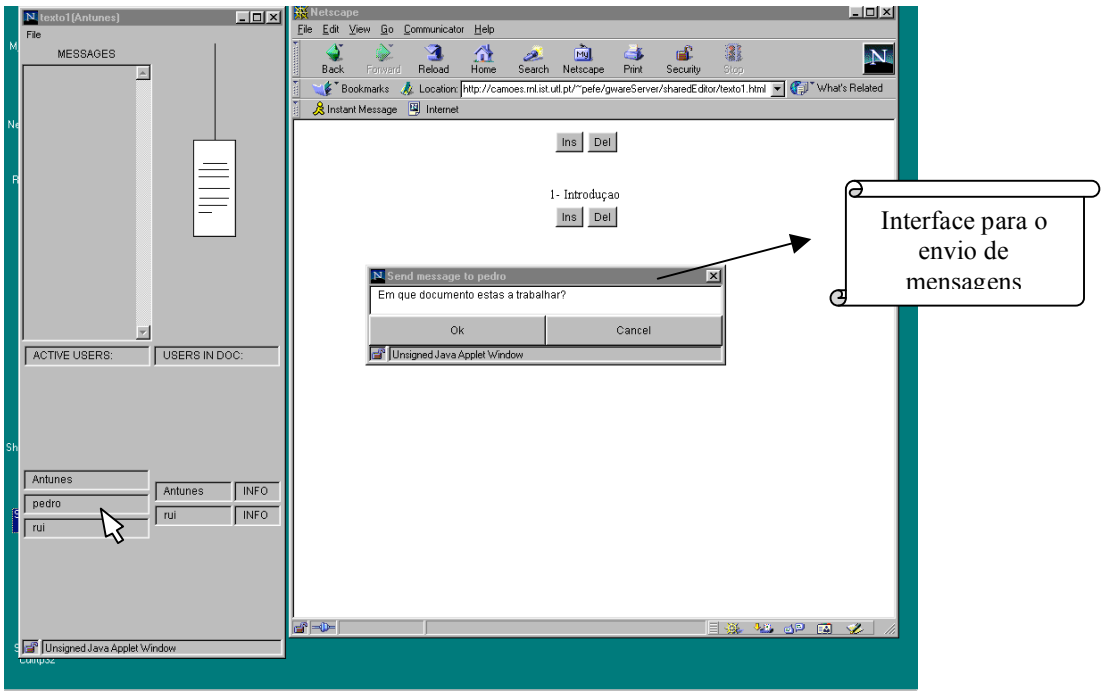
Após entrar no documento, o utilizador depara-se com uma situação semelhante a esta:



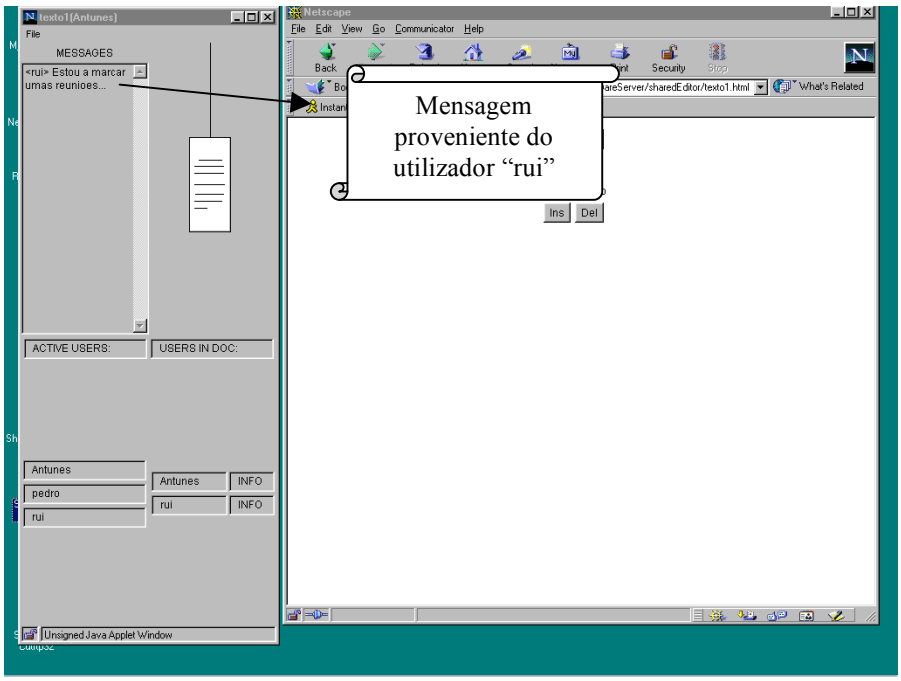
Ao passar o cursor sobre a caixa "info", o cliente *webmonitor* contacta com o Servidor para lhe fornecer informação actualizada sobre o utilizador respectivo.



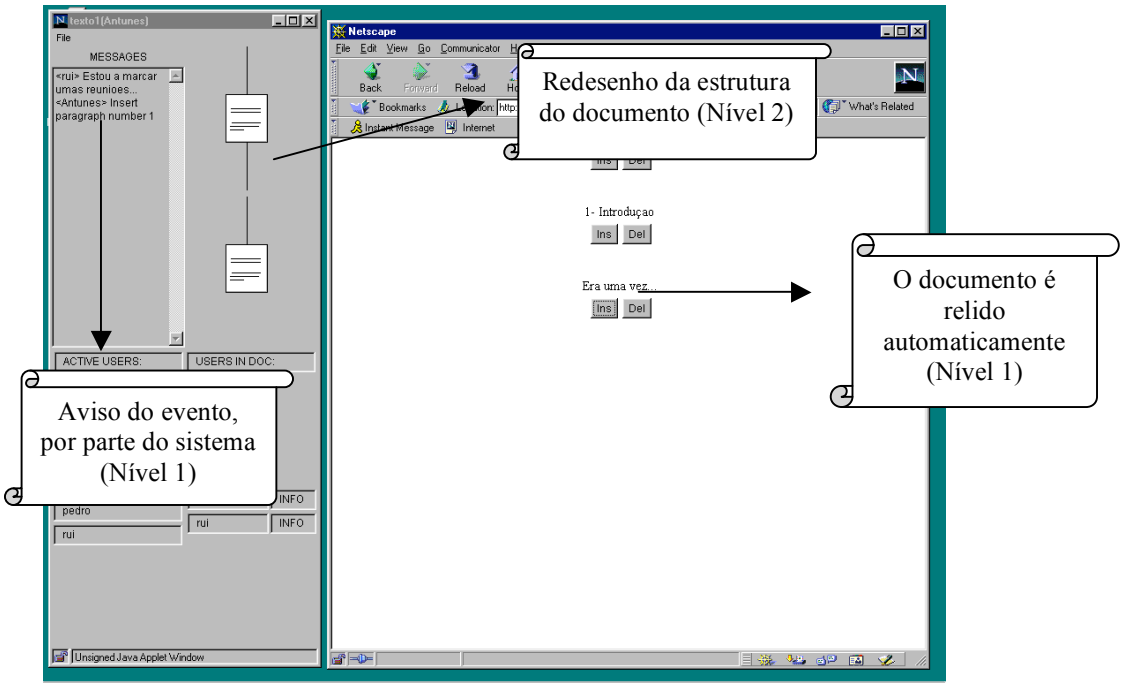
Quando se prime no botão esquerdo do rato sobre o nome de um utilizador, uma janela de diálogo aparece para se enviar uma mensagem ao mesmo:



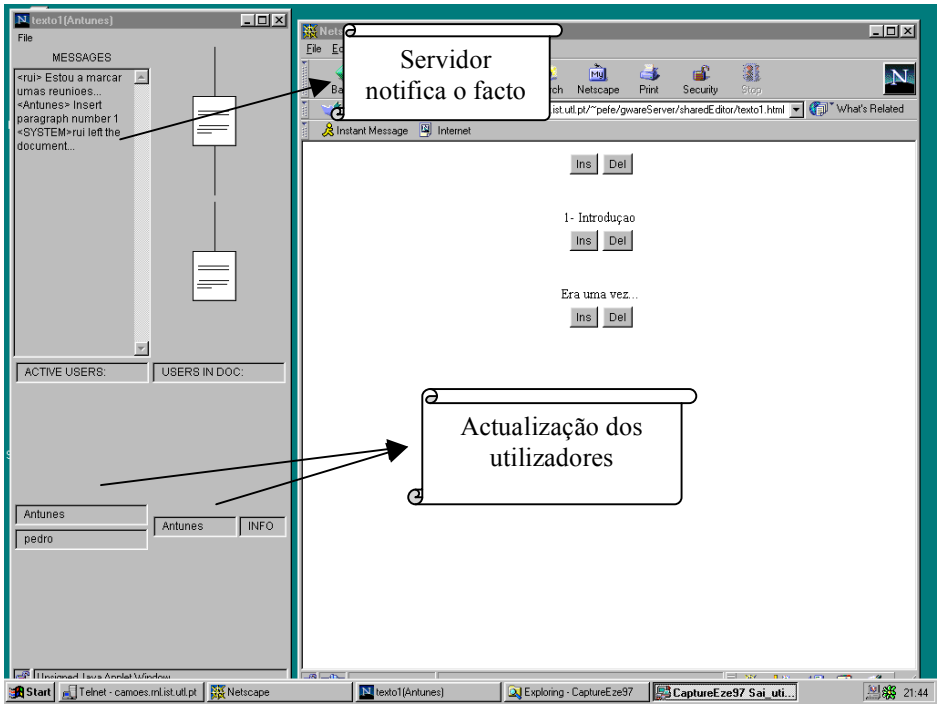
Nesta figura, pode-se visualizar a recepção de uma mensagem vinda de outro utilizador:



Acontecimentos após a inserção de um novo elemento no documento, consoante o nível de interacção (ver cap. 2.1):



Sinalização da saída de um utilizador:



9 – Referências

[1] G. Abowd, R. Beale, A. Dix e J. Finlay. *Human Computer Interaction*. Prentice Hall. 1993

[2] W. Appelt, R. Bentley, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor e G. Woetzel, *Basic Support for Cooperative Work on the World Wide Web.* International Journal of Human Computer Studies of the WWW. Spring 1997. Academic Press. Cambridge.

[3] K. Arnold e J. Gosling. *The Java Programming Language*. Addinson Wesley. 1ª Edição Maio 1996.

[4] S. Greenberg e M. Roseman .(in press) *Groupware Toolkits for Synchronous Work*. In M. Beaudouin-Lafon, editor Computer-Supported Cooperative Work, Trens in Software Series, John Wiley & Suns Ltd.

[5] K. Palfreyman e T. Rodden, *A Protocol for User Awarnes on the World Wide Web*. ACM 1996 Conference on (CSCW'96) Computer Supported Cooperative Work. 1996.

[6] P. Ferreira. *Acetatos das aulas teóricas da disciplina de Arquitecturas de Redes de Grande Escala*. 1997.

[7] V. Quercia e S. Spainborn. *Webmaster in a Nutshell*. O'Relly 1ª Edição Outubro 1996.

[8] D. Wooley, *Choosing Web Conferencing Software*. International University Consortium Conference in WWW Course Development & Delivery. 1996.

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

[10] *World Wide Web Consortium* http://www.w3c.org

[11] BSCW,URL:http://bscw.gmd.de