INSTITUTO
SUPERIOR
TÉCNICO

SUPORTE E CONTROLO DA INTERACÇÃO EM SISTEMAS DE TRABALHO COOPERATIVO

Relatório do Trabalho Final de Curso
Licenciatura em Engenharia Informática e de Computadores

Carlos Manuel Gregório Alves, Nº 38390
Daniel Pereira Marques da Silva, Nº 38398

Setembro de 1997

SUPORTE E CONTROLO DA INTERACÇÃO EM SISTEMAS DE TRABALHO COOPERATIVO

Realizado por:

Carlos Manuel Gregório Alves, Nº 38390
Daniel Pereira Marques da Silva, Nº 38398

Sob a orientação de:

Prof. Pedro Alexandre de Mourão Antunes
Eng.º António Rito da Silva

Relatório Final do Trabalho Final de Curso

LICENCIATURA EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES
INSTITUTO SUPERIOR TÉCNICO - UNIVERSIDADE TÉCNICA DE LISBOA

INESC - INSTITUTO DE ENGENHARIA DE SISTEMAS E COMPUTADORES

Setembro de 1997

RESUMO

Este trabalho estuda e demonstra mecanismos de suporte à interacção em sistemas de trabalho cooperativo em que um grupo de utilizadores manipula simultaneamente um espaço de dados comum. O tipo de sistemas identificado apresenta uma riqueza de interacção que os distingue dos sistemas mono-utilizador, levantando a necessidade de introduzir mecanismos de interface utilizador e de estruturação da interacção adequados à partilha de dados. A utilização destes, introduz problemas de controlo de concorrência que devem ser evitados e/ou resolvidos.

À luz destes problemas descreve-se um modelo adequado para estruturação da interacção em grupo, que apresenta um conjunto de mecanismos simples para manipulação de dados partilhados através da interface utilizador. Este, é suficientemente genérico e adaptável, de modo a construir facilmente famílias de aplicações baseadas num conjunto de técnicas de interacção comuns. O modelo é enquadrado num conjunto de fundamentos teóricos e de trabalhos já realizados neste domínio.

Finalmente, descreve-se uma biblioteca que implementa o modelo e uma aplicação facilitadora de geração de ideias em grupo que a instancia, exemplificando e validando as conclusões retiradas do estudo da classe de sistemas em causa.

Para a Alice, para o Carlos e para a Sílvia

Para a Lídia, para o Armando, para a Sóninha e para o Pedro

AGRADECIMENTOS

O nosso primeiro agradecimento dirige-se ao nossos orientadores, Prof. Pedro Antunes e Eng.º António Rito da Silva pelo apoio e interesse demonstrado, bem como pelas críticas e sugestões apresentadas. O trabalho aqui apresentado só foi possível conjugando as suas visões complementares sobre um mesmo problema. Obrigado também, por nos terem dado uma liberdade criativa que dificilmente voltaremos a encontrar no nosso futuro profissional.

Um agradecimento especial para o Grupo de Engenharia de *Software* do Inesc, onde encontrámos um agradável ambiente de trabalho, e que nos disponibilizou os meios necessários à realização deste trabalho. Em particular aos professores da sala 532 que nos deixaram "invadir" o seu local de trabalho na recta final.

Às Eng.as Tânia Ho e Isabel Soares que por nos cederem o seu trabalho, contribuíram decisivamente para a concretização daquele que aqui apresentamos. Uma menção também para os Eng.os Vasco Paulo, Luís Gil e João Martins que realizaram um conjunto de ferramentas que se revelaram fundamentais para concretizar as nossas ideias.

Aos nossos Pais e Irmãos pelo apoio, carinho e paciência demonstrados ao longo de todos estes anos.

À Ana, à Ana, ao Carlos, ao Hugo, ao João, ao Jorge, à Leonor, ao Lúcio, ao Miguel, ao Miguel, ao Miguel, ao Michel, ao Paulo (Gordo), ao Paulo (Gordo), ao Paulo, ao Pedro, ao Rui, à Sílvia, à Sandra, à Sandra, ao Toni (camarada de tantas "guerras") e ao Zé por estarem sempre connosco partilhando preocupações e muitas alegrias.

Lisboa, Setembro de 1997

Carlos Manuel Gregório Alves

Daniel Pereira Marques da Silva

ÍNDICE

1. INTRODUÇÃO **1**

1.1 MOTIVAÇÃO 1
1.2 PROBLEMAS 1
1.3 OBJECTIVO 2
1.4 RESULTADOS 2
1.5 CONTEXTO 2
1.6 ESTRUTURA DO RELATÓRIO 2

PARTE I - ENQUADRAMENTO

2. FUNDAMENTOS **5**

2.1 SISTEMAS DE TRABALHO COOPERATIVO 5
2.1.1 Taxionomia *6*
2.2 SUPORTE E CONTROLO DA INTERACÇÃO EM GRUPO 6
2.2.1 Partilha de dados *7*
2.2.2 Interface utilizador *7*
2.2.2.1 Modelos de espaço público 8
2.2.2.2 Monitorização 8
2.2.3 Estruturação da interacção *8*
2.2.4 Arquitectura *8*

3. TRABALHO RELACIONADO **11**

3.1 SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES 11
3.1.1 Partilha de dados *11*
3.1.1.1 Framework para geração e controlo de concorrência 12
3.1.1.1.1 Padrão de sincronização de objecto 13
3.1.1.1.2 Padrão de recuperação de objecto 14
3.1.1.1.3 Exemplo de composição: Composição passiva síncrona recuperável 15
3.1.1.1.4 A framework no suporte de aplicações de trabalho cooperativo 15
3.1.2 Estruturação da interacção *17*
3.1.2.1 EGRET 17
3.1.3 Interface utilizador *17*
3.1.3.1 EdGar 18
3.2 APLICAÇÕES 18
3.2.1 NGTool e NGMeeting *18*
3.2.1.1 Desenho das aplicações 19
3.2.1.1.1 Estruturação da interacção 19
3.2.1.1.2 Interface utilizador 19
3.2.1.1.3 Partilha de dados 20
3.2.1.2 Realização 20
3.2.1.2.1 MBus 21
3.2.2 GroupSystems *21*

PARTE II - MODELO DE ESTRUTURAÇÃO DA INTERACÇÃO

4. SUPORTE À INTERACÇÃO **25**

4.1 INTRODUÇÃO 25
4.2 OBJECTOS 25
4.2.1 Contentores e Conteúdos *25*
4.2.1.1 Propriedades dinâmicas 26
4.2.2 Conexões *26*
4.2.3 Monitores *28*
4.2.4 Resumo *29*
4.3 OPÇÕES DE DESENHO 29
4.3.1 Interface utilizador *29*
4.3.1.1 Representação de Contentores 29
4.3.1.2 Apresentação de Conteúdos 30
4.3.1.3 Monitorização 30
4.3.2 Estruturação da interacção *32*

5. CONTROLO DA INTERACÇÃO 33

5.1 INTRODUÇÃO 33
5.2 ESTRUTURAÇÃO DA INTERACÇÃO 33
5.3 OBJECTOS 34
5.3.1 Controlo de operações de edição *34*
5.3.2 Controlo de operações de estruturação *35*
5.4 INTERFACE UTILIZADOR 36
5.4.1 Monitorização da concorrência na edição de conteúdos simples *36*
5.4.2 Monitorização da concorrência na estruturação de Contentores *36*

PARTE III - REALIZAÇÕES

6. REALIZAÇÃO DO MODELO DE INTERACÇÃO 39

6.1 MODELO DE OBJECTOS DE SUPORTE 39
6.2 MODELO DE OBJECTOS DE CONTROLO 41

7. EXEMPLO DE APLICAÇÃO *IDEAGEN* 43

7.1 ENQUADRAMENTO 43
7.2 REQUISITOS 43
7.3 DESENHO 44
7.3.1 Funcionalidades do IDEAGEN *44*
7.3.1.1 Geração e manipulação de ideias 44
7.3.1.2 Talker 45
7.3.1.3 Confirm 46
7.3.2 Arquitectura *46*
7.3.3 Modelos de objectos *47*
7.3.3.1 IDEAGEN 47
7.3.3.1.1 Instanciação do modelo de suporte à interacção 48
7.3.3.1.2 Contentores 49
7.3.3.1.3 Conteúdos simples 50
7.3.3.2 IDEASERVER 50
7.3.4 Captura de acções de interacção *52*

8. CONCLUSÕES E TRABALHO FUTURO 53

BIBLIOGRAFIA 57

APÊNDICE A - RESUMO DA NOTAÇÃO UML 61

APÊNDICE B - EXEMPLOS DE UTILIZAÇÃO 63

B.1 INTERACÇÕES DISPONIBILIZADAS A UM UTILIZADOR 63

B.2 Distribuição e relaxação da coerência visual....66
B.3 Controlo da concorrência....69

APÊNDICE C - IMPLEMENTAÇÃO EM C++ 71

C.1 Suporte à interacção 71
C.2 Controlo da interacção 74
C.3 Instanciação no *IdeaGen*....75

LISTA DE FIGURAS

Figura 1 - Taxionomia espaço-tempo de sistemas de trabalho cooperativo....6
Figura 2 - Suporte à interacção em grupo....7
Figura 3 - Arquitectura da *framework*....12
Figura 4 - Padrão de sincronização de objecto....13
Figura 5 - Padrão de recuperação de objecto....14
Figura 6 - Composição passiva síncrona recuperável....15
Figura 7 - Situação de conflito....15
Figura 8 - Arquitectura de dados para desenvolvimento experimental de soluções para controlo da concorrência....16
Figura 9 - Interface gráfica do *NGTool*....20
Figura 10 - Controlo da concorrência (O utilizador da esquerda requer o trinco que é detido pelo utilizador da direita)20
Figura 11 - Módulos da aplicação *NGMeeting*....21
Figura 12 - Exemplo de utilização do *GroupSystems*....22
Figura 13 - Hierarquia de propriedades de cada Contentor....26
Figura 14 - Afectação dos objectos envolvidos na fase de Iniciação....27
Figura 15 - Afectação dos objectos envolvidos na fase de Finalização....27
Figura 16 - Matriz de componentes dos objectos do modelo....29
Figura 17 - Representações de Contentores....29
Figura 18 - Apresentação de um Conteúdo simples....30
Figura 19 - Apresentações alternativas de Conteúdos compostos....30
Figura 20 - Monitorização da propriedade de manipulação dos Contentores....31
Figura 21 - Monitorização da propriedade de manipulação dos Conteúdos simples....31
Figura 22 - Monitorização de alterações aos Conteúdos num modo *WYSIWIMS*....31
Figura 23 – Monitorização do estado de visualização dos Conteúdos Compostos....31
Figura 24 - Monitor de elasticidade temporal....31
Figura 25 – Estados para edição concorrente de Conteúdos simples....34
Figura 26 – Detecção de conflitos em operações de estruturação de dados....35
Figura 27 – Estados do monitor de concorrência....36
Figura 28 - Modelo de objectos de suporte à interacção....39
Figura 29 - Diagrama de Sequência....40
Figura 30 - Controlo de concorrência na edição de Conteúdos....42
Figura 31 – Interface gráfica do IDEAGEN....45
Figura 32 - Arquitectura conceptual....46
Figura 33 - Arquitectura real....47
Figura 34 - Núcleo de objectos do IdeaGen....48
Figura 35 - Instanciação do modelo de suporte à interacção no IdeaGen....49
Figura 36 - Contentores para geração e manipulação de ideias....49
Figura 37 - Contentores para geração e manipulação de mensagens do *Talker*....50
Figura 38 - Contentores para geração e manipulação de mensagens de *Confirm*....50
Figura 39 - Outros Contentores com funcionalidades diversas....50
Figura 40 – Diagrama de classes do IDEASERVER....51
Figura 41 - Modelação da interacção a partir do rato....52
Figura 42- Conexão entre uma Origem durável e o espaço livre (I)....63
Figura 43 - Conexão entre uma Origem durável e o espaço livre (II)....63
Figura 44 - Conexão entre uma Origem transitória e o espaço livre (I)....64
Figura 45 - Conexão entre uma Origem transitória e o espaço livre (II)....64
Figura 46 - Conexão entre uma Origem transitória e outro Contentor (I)....64
Figura 47 - Conexão entre uma Origem transitória e outro Contentor (II)....64
Figura 48 - Conexão entre um Contentor durável de Conteúdo simples durável e outro Contentor (I)....65
Figura 49 - Conexão entre um Contentor durável de Conteúdo simples durável e outro Contentor (II)....65
Figura 50 - Conexão entre um Contentor durável de Conteúdo simples transitório e outro Contentor (I)....65

Figura 51 - Conexão entre um Contentor durável de Conteúdo simples transitório e outro Contentor (II)....................66
Figura 52 - Conexão entre um Contentor transitório de Conteúdo simples transitório e outro Contentor (I) 66
Figura 53 - Conexão entre um Contentor transitório de Conteúdo simples transitório e outro Contentor (II)...............66
Figura 54 - Conexão que não altera a estrutura hierárquica dos Contentores..67
Figura 55 - Conexão que altera a estrutura hierárquica dos Contentores 67
Figura 56 - Monitorização da edição de Conteúdo simples (a) 68
Figura 57 - Monitorização da edição de Conteúdo simples (b)...68
Figura 58 - Monitorização da edição de Conteúdo simples (c) 68
Figura 59 - Monitorização da edição de Conteúdo simples (d)...68
Figura 60 - Edição de dois Conteúdos simples em simultâneo (a) 69
Figura 61 - Edição de dois Conteúdos simples em simultâneo (b)...69
Figura 62 - Edição de dois Conteúdos simples em simultâneo (c) 69
Figura 63 - Edição de dois Conteúdos simples em simultâneo (d)...70
Figura 64 – Manipulação de Contentores (a)...70
Figura 65 – Manipulação de Contentores (b) 70
Figura 66 – Manipulação de Contentores (c)...70
Figura 67 - Classe Container 71
Figura 68 – Métodos de iniciação como Origem e finalização como Destino 72
Figura 69 - Classe Content 72
Figura 70 - Classe Connection e seus métodos de iniciação e finalização 73
Figura 71 - Controlo da Concorrência na edição de Conteúdos simples...74
Figura 72 – Instanciação de Container no *IDEAGEN* 75
Figura 73 - Métodos particulares à aplicação para abertura e fecho da representação do Contentor................................76
Figura 74 - Instanciação de Connection para a aplicação *IDEAGEN* 76

Capítulo 1

INTRODUÇÃO

É dado um enquadramento do trabalho num conjunto de problemas que obstam à construção de sistemas de trabalho cooperativo interactivos e é dada uma panorâmica geral sobre os resultados obtidos na sua solução.

1.1 Motivação

Na sociedade actual, a comunicação e cooperação entre as pessoas torna-se uma necessidade cada vez mais premente. Neste domínio, os computadores são cada vez mais utilizados, proporcionando novas e diferentes formas de interacção suportadas por inovações tecnológicas.

Os caminhos que hoje em dia são percorridos, apontam para a construção de sistemas organizacionais que integrem as capacidades de processamento da informação e suporte às capacidades de comunicação e cooperação. Neste contexto surge um novo esforço interdisciplinar: a construção de sistemas de trabalho cooperativo (CSCW - *Computer Supported Cooperative Work*) [Ellis 91]. Este esforço envolve especialistas de áreas diversificadas, desde a Ciência dos Computadores até à Sociologia.

Os novos sistemas desenvolvidos neste âmbito diferenciam-se dos sistemas mono-utilizador por envolverem as diferentes formas de interacção entre os membros de um grupo, para além das habituais técnicas de interacção pessoa-máquina.

Exemplos de sistemas de trabalho cooperativo são: editores de documentos, sistemas de suporte à decisão em grupo, sistemas de controlo de fluxos de trabalho, ambientes de programação em equipa, etc.

Entre estes sistemas destacam-se aqueles em que os utilizadores interagem em tempo-igual, ou seja, onde realizam actividades simultâneas sobre um contexto partilhado. Esta classe de sistemas assume especial relevância dadas as necessidades actuais de comunicação e cooperação.

1.2 Problemas

No contexto introduzido, são três os problemas identificados que requerem um solução integrada:

1. Os actuais sistemas de trabalho cooperativo apresentam técnicas de **suporte à interacção** pouco adequadas à manipulação simultânea de dados por grupos de pessoas.
2. As aplicações tradicionais em que diversos computadores operam sobre um espaço de dados comum, apresentam situações de conflito na gestão destes. A forma de os resolver tem sido largamente estudada, testada e divulgada pelos modernos sistemas operativos. Estas técnicas recorrem a mecanismos de **controlo da concorrência** baseados em considerações exclusivamente tecnológicas. No caso dos sistemas de trabalho cooperativo, a interacção pessoa-máquina e a interacção em grupo são também geradoras de conflitos que devem ser resolvidos [Antunes 94] [Greenberg 94].

3. Por último, a resolução dos problemas já identificados esbarra em considerações exclusivamente tecnológicas, mais concretamente na dificuldade actual no **suporte computacional para aplicações multimédia cooperativas distribuídas** [Rodden 92]. A capacidade dos actuais sistemas não permite introduzir todas as facilidades de comunicação desejáveis, nomeadamente em redes de grande escala.

1.3 Objectivo

No contexto do presente trabalho pretendem-se apresentar soluções para os dois primeiros problemas à luz do terceiro, ou seja:

Definição de um modelo integrado para suporte e controlo da interacção em grupo, adequado a redes de grande escala.

1.4 Resultados

O objectivo definido foi atingido pela definição do modelo pretendido. Este baseia-se na manipulação de objectos gráficos elementares e na sua estruturação, partilha e manipulação através de um conjunto simples de operações.

O modelo referido foi materializado numa biblioteca para suporte ao desenvolvimento da classe de aplicações identificada. Foi ainda construído um protótipo que instancia o modelo para sua exemplificação.

1.5 Contexto

Este trabalho foi realizado no âmbito de dois grupos de investigação da estrutura do INESC (Instituto de Engenharia de Sistemas e Computadores):

- **Grupo de Engenharia de *Software* :** enquadrado no projecto DASCo (*Development of Distributed Applications with Separation of Concerns*), mais especificamente na utilização de produtos assistentes nas actividades de desenvolvimento do aspecto de controlo da concorrência. Neste contexto foi utilizada uma *framework* desenvolvida para o efeito;
- **Grupo de Técnicas de Interacção e Multimédia :** integrado na investigação em interfaces de sistemas cooperativos. O protótipo desenvolvido foi baseado no trabalho já desenvolvido no teste de técnicas de facilitação de reuniões distribuídas.

1.6 Estrutura do Relatório

O presente relatório encontra-se estruturado em três partes fundamentais:

I. **Enquadramento :** São apresentados os conceitos básicos que permitem a compreensão do trabalho desenvolvido acompanhados de uma análise dos problemas identificados e avançadas soluções gerais para a sua solução. São também apresentados os sistemas que serviram de suporte à realização do protótipo final.

II. **Modelo de interacção :** É definido o modelo proposto nas suas componentes de suporte e controlo da interacção.

III. **Realizações :** São descritos os sistemas desenvolvidos, apresentando inicialmente a implementação do modelo, seguida de uma discussão e ilustração da sua instanciação no protótipo.

Finalmente, são apresentadas conclusões e caminhos de trabalho futuro.

Parte I

ENQUADRAMENTO

Capítulo 2

FUNDAMENTOS

Neste capítulo é feita uma introdução aos conceitos envolvidos no trabalho. Não se pretende fazer uma descrição exaustiva, apenas permitir a compreensão dos Capítulos seguintes. Abordagens mais completas podem ser encontradas nas referências indicadas ao longo do texto. A apresentação dos conceitos é acompanhada de uma análise preliminar de soluções para os problemas identificados.

2.1 Sistemas de trabalho cooperativo

Num sentido lato, o trabalho presente enquadra-se na área multi-disciplinar dos sistemas de trabalho cooperativo. Nela, são englobados aspectos tecnológicos, mas também aspectos sociológicos e psicológicos. O seu objectivo é o desenvolvimento de sistemas que permitam a partilha de informação e criação de condições para trabalhar em grupo de forma eficiente [Ellis 91].

Actualmente, a maioria dos sistemas informáticos suportam uma única forma de interacção, a interacção pessoa-máquina típica dos sistemas operativos de tempo-partilhado onde a interacção dos utilizadores, apesar de partilharem uma mesma máquina, é propositadamente isolada dos restantes. Ou seja, o sistema é visto como sendo exclusivo de cada indivíduo. No entanto, tal é paradoxal, uma vez que a maior parte das suas actividades ocorrem em grupo. Tais sistemas actuam com uma filosofia contrária ao que seria natural.

O aspecto apontado no parágrafo anterior implica um esforço conjunto que permita redefinir os aspectos envolvidos neste tipo de sistema. Estes são estudados nas seguintes áreas de investigação:

- **Sistemas Distribuídos :** A natureza dos sistemas é eminentemente distribuída. Os utilizadores encontram-se dispersos pelos seus postos de trabalho. Nesta área torna-se necessário adaptar as técnicas tradicionais para este novo tipo de sistemas. Destacam-se no âmbito deste trabalho os aspectos de replicação e controlo da concorrência;
- **Redes de Comunicações :** São introduzidas novas necessidades de comunicação efectiva e diversificada. Fornece o suporte para a utilização de aplicações multimédia distribuídas em tempo real;
- **Interfaces Pessoa-Máquina :** Os modelos tradicionais de interacção têm necessariamente de ser estendidos para suporte de vários utilizadores e incorporar os processos de interacção em grupo;
- **Inteligência Artificial :** De forma a automatizar alguns dos processos de cooperação torna-se interessante introduzir agentes inteligentes que exibam um comportamento social aceitável;
- **Sociologia :** Neste campo devem ser considerados os processos habituais de cooperação em grupo e de monitorização de presença e actividade dos seus membros.

No trabalho desenvolvido foram apenas considerados aspectos nas áreas de Sistemas Distribuídos e Interfaces Pessoa-Máquina. Foram ainda testados alguns mecanismos de monitorização de actividades dos membros do grupo.

2.1.1 Taxionomia

Uma vez indicados os aspectos a abordar, interessa agora restringir a classe dos sistemas a tratar. Para tal, apresenta-se uma taxionomia de sistemas de trabalho cooperativo em duas vertentes: espaço e tempo (ver Figura 1). Descrições mais pormenorizadas acompanhadas de exemplos podem ser encontradas em [Ellis 91] [Rodden 92] e [Antunes 96b].

- **Espaço :** Os membros do grupo podem encontrar-se num mesmo espaço físico ou distribuídos por espaços onde não mantêm contacto visual.
- **Tempo :** Dependendo da tarefa a realizar, um grupo tem ou não necessidade de interactuar simultaneamente. A interacção em tempo-igual introduz um contexto partilhado comum entre os membros do grupo, onde estes interagem. Por esta razão definem-se como convergentes.

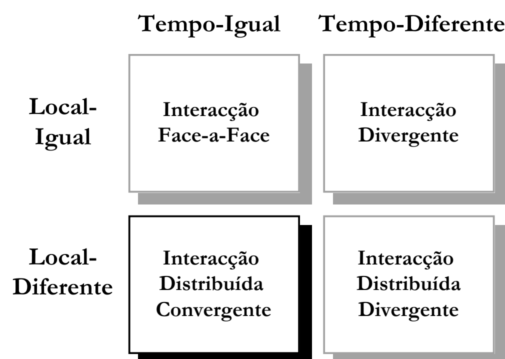


Figura 1 - Taxionomia espaço-tempo de sistemas de trabalho cooperativo

Na Figura 1 destacam-se os sistemas de trabalho cooperativos onde a interacção é distribuída convergente, por duas razões fundamentais: respondem a uma grande parte das necessidades de comunicação e cooperação e por englobarem e maximizarem o conjunto das funcionalidades introduzidas pela generalidade dos sistemas de trabalho cooperativo.

Uma vez descrito o espectro de sistemas e áreas envolvidas, considera-se que o trabalho desenvolvido e o restante relatório incide sobre os aspectos de **suporte e controlo da interacção distribuída convergente em sistemas de trabalho cooperativo**. Estes aspectos serão desenvolvidos mais em pormenor nas duas secções seguintes.

2.2 Suporte e controlo da interacção em grupo

São vários os modelos já estudados que permitem efectivar a interacção entre grupos de pessoas, bem como de técnicas facilitadoras que estruturam a cooperação [Antunes 96b]. Interessa agora, apresentar técnicas e mecanismos que permitam suportar os diferentes aspectos envolvidos.

De um modo geral, o suporte à interacção em grupo é composto por **partilha da dados, estruturação da interacção e interface utilizador**.

A Figura 2 apresenta o modelo geral (independente da arquitectura) de suporte à interacção em grupo, onde se podem observar as suas três componentes indicadas.

No presente trabalho a preocupação centra-se na estruturação da interacção e na forma como esta afecta as duas outras.

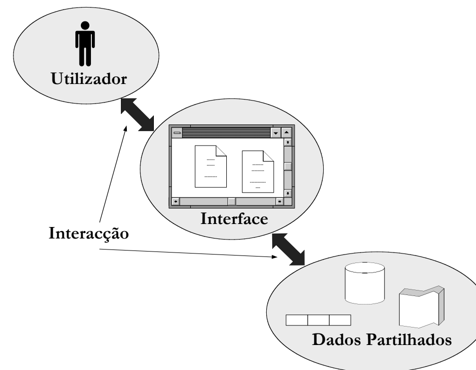


Figura 2 - Suporte à interacção em grupo

2.2.1 Partilha de dados

A partilha da dados permite definir um contexto de trabalho comum entre os indivíduos. Esta funcionalidade implica a existência de um mecanismo de coerência de dados, especialmente em arquitecturas de dados replicados. O controlo na partilha é feito recorrendo a mecanismos de controlo da concorrência. Várias têm sido as propostas apresentadas para a resolução deste problema [Marques 90] [Barghouti 91]. Distinguem-se duas classes fundamentais, sendo permitidos graus intermédios de coerência:

- **Controlo pessimista**. O controlo do acesso aos objectos de dados é feito antes da manipulação.
- **Controlo optimista**. Permite fazer a manipulação dos objectos sem qualquer restrição. O controlo é feito depois de manipulados os dados, sendo admitidos conflitos. A resolução destes pode ser feita pelo sistema, ou por ele promovida junto dos utilizadores.

São também admitidas variações onde se efectue controlo antes e depois do acesso, bem como onde não exista controlo de todo.

Na classe de sistemas em estudo, uma parte considerável das interacções são duradouras, isto é, dão-se à velocidade de trabalho dos utilizadores, em oposição ao requisito de atomicidade habitualmente associado às transacções distribuídas [Grey 93]. Para este tipo de interacção devem ser disponibilizadas técnicas de controlo optimista da concorrência, podendo os conflitos ser resolvidos pelos utilizadores ou pelas aplicações. A resolução destes conflitos pode ser através de mecanismos de recuperação.

O controlo deve ser acompanhado de mecanismos de monitorização (ver 2.2.2.2).

2.2.2 Interface utilizador

A interface utilizador é responsável por mediar o acesso aos dados, reflectir os processos de interacção em grupo e manipulação das estruturas de dados partilhadas. Para realizar tal tarefa torna-se necessário definir um espaço virtual através do qual os diversos utilizadores podem interactuar entre si e com os dados a manipular. Estes denominam-se **espaços públicos**.

Nos processos de trabalho cooperativo, existem situações onde se torna possível e até necessário executar trabalho próprio num contexto privado. É pois possível, introduzir a noção de **espaço privado** de modo a suportar a particularidade de alguns modelos de interacção. Uma discussão das razões referidas pode ser encontrada em [Antunes 94].

A necessidade de evitar conflitos pode implicar restrições à interacção. Estas restrições estão intimamente relacionadas com o modelo de interacção escolhido e com a política de controlo da concorrência.

2.2.2.1 Modelos de espaço público

Ao nível da interface utilizador deverão ser utilizadas janelas gráficas para implementar o conceito de espaço público de forma distribuída. Esta manipulação pode ser feita com diferentes graus de coerência entre as vistas sobre os dados partilhados [Antunes 94]. Destacam-se três:

- ***WYSIWIS* (*What You See Is What I See*)**. Requer que todos os atributos dos espaços de dados e visual sejam iguais para os diferentes utilizadores. Implementa a coerência estrita de interface e de dados partilhados;
- ***WYSIWIMS* (*What You See is What I Might See*)**. Permite vistas independentes sobre os mesmos dados. Relaxa a coerência da interface apresentada a cada utilizador;
- ***WYGIWIG* (*What You Get Is What I Get*)**. Permite visualizar estados intermédios em que os utilizadores divergem na manipulação dos dados, devendo estes ser consciencializados dessa situação. Relaxa a coerência na partilha de dados. É mais facilmente associado à utilização de mecanismos de controlo optimista da concorrência.

2.2.2.2 Monitorização

A interacção recorrente da actividade em grupo apresenta um espectro bastante rico de formas de comunicação, como sejam: voz, expressões faciais, gestos, etc. Estes mecanismos enriquecem e facilitam a cooperação. Dadas as dificuldades de cariz tecnológicas inicialmente identificadas, torna-se necessário enriquecer a interface utilizador de mecanismos que permitam a cada membro do grupo tomar consciência das actividades e da presença dos restantes. A monitorização pode cobrir vários aspectos da interacção e pode implicar uma convergência maior ou menor dos utilizadores para actividades que estão a ser realizadas. A monitorização deverá incidir nos aspectos de manipulação da interface e de manipulação dos dados. Deverá ainda adequar-se e acompanhar as técnicas de relaxação visual e de dados, bem como o controlo da interacção.

Um maior grau de convergência é normalmente associado a modos de interacção do tipo *WYSIWIS*.

2.2.3 Estruturação da interacção

A estruturação da interacção entre utilizadores e a manipulação e organização dos dados partilhados pode ser definida ao nível das aplicações. No entanto, há que ter em conta que o modo de estruturação dos dados influencia-a directamente. Consequentemente o controlo da concorrência e a interface utilizador são também afectados.

A escolha da forma de estruturação influencia decisivamente a escolha das técnicas de interacção. Destaca-se a organização hipertexto [Balasubramanian 93], onde a informação é organizada em diversos nós que se encontram relacionados por ligações. Este tipo de organização permite fragmentar os dados, minimizando a ocorrência de conflitos na sua manipulação. Suporta os modos *WYSIWIMS* e *WYGIWIG*.

Por outro lado, a adaptação da semântica das aplicações a uma organização deste tipo permite criar conjuntos de aplicações para diferentes domínios, com mecanismos idênticos de estruturação da interacção e de interface utilizador. Este resultado apresenta vantagens por permitir ao utilizador criar habituação, facilitando a aprendizagem. No entanto, o desenho deste tipo de sistemas deve ser cuidado de forma a não introduzir aspectos negativos ao nível da interface utilizador. Uma discussão destes aspectos pode ser encontrada em [Balasubramanian 93].

2.2.4 Arquitectura

A partilha de informação e a interacção influenciam de forma directa a arquitectura das aplicações. Consideram-se três propostas fundamentais de arquitectura [Antunes 96b]:

- **Centralizada mono-utilizador**. Um servidor integra os dados a manipular, bem como a informação visual da aplicação, sendo esta adquirida e distribuída por diferentes canais de entrada e saída. É obrigatoriamente associado ao modo *WYSIWIS*;
- **Centralizada multi-utilizador**. Os dados visuais são distribuídos, mantendo-se os dados a manipular num nó servidor central. Ambas as arquitecturas centralizadas encorajam a utilização de mecanismos de controlo pessimista da concorrência. Permite a utilização do modo *WYSIWIMS*;
- **Distribuída**. Tanto os dados visuais como os dados a manipular são distribuídos entre os diversos nós. A coerência de dados pode ser obtida utilizando mecanismos transaccionais distribuídos [Gray 93]. Encoraja a utilização de mecanismos de controlo optimista de concorrência e dos modos *WYSIWIMS* e *WYGIWIG*.

Uma descrição mais fundamentada da composição das características apresentadas pode ser consultada em [Antunes 94] ,[Antunes 96b] e [Greenberg 94].

Capítulo 3

TRABALHO RELACIONADO

Este capítulo apresenta trabalho relacionado em diferentes aspectos da realização de sistemas de trabalho cooperativo distribuídos convergentes. Apresentam-se ferramentas de suporte ao desenvolvimento e aplicações existentes que de alguma forma se relacionam com o estudo feito até aqui. Especial destaque é dado ao NGMeeting e à framework de concorrência por terem servido de suporte às realizações práticas apresentadas na terceira parte deste relatório.

3.1 Suporte ao desenvolvimento de aplicações

A conclusão do Capítulo anterior apresenta um elevado grau de complexidade no desenho e realização das aplicações em causa. Torna-se pois necessário, apresentar metodologias e ferramentas que de algum modo auxiliem o seu desenvolvimento. Em seguida são introduzidos alguns estudos que se centram nas componentes identificadas na Figura 2.

3.1.1 Partilha de dados

O desenvolvimento da componente de partilha de dados pode ser feito segundo as metodologias de desenvolvimento habituais [Pressman 91]. Neste contexto apresenta-se uma abordagem proposta em [Silva 95] que realça a separação de aspectos no desenvolvimento de aplicações distribuídas. Esta metodologia foi escolhida de forma a enquadrar a utilização da *framework* de concorrência apresentada em 3.1.1.1.

Os aspectos identificados são: partição, replicação, gestão de nomes, concorrência, falhas, configuração e comunicação. É proposta uma metodologia de desenvolvimento construtiva, permitindo a verificação parcial de resultados.

A metodologia segue seis passos :

1. **Análise.** Definição dos requisitos da aplicação e seu enquadramento nos aspectos considerados.
2. **Versão não distribuída.** É desenhada uma aplicação ignorando os aspectos relacionados com a distribuição.
3. **Distribuição lógica.** A aplicação é redesenhada como um conjunto de mundos que conseguem comunicar através da troca de mensagens.
4. **Persistência.** Enriquece-se a aplicação com o suporte à persistência, adaptando os aspectos de replicação e gestão de nomes.
5. **Concorrência.** Integra-se a geração e controlo da concorrência.
6. **Robustez.** Depois desta fase a aplicação deverá ser tolerante a faltas.
7. **Distribuição física.** Transforma-se a versão lógica em física implementando os mecanismos concretos de cada plataforma a partir dos mecanismos abstractos definidos em 3.

Cada fase do processo atribui uma actividade à refinação de cada aspecto, sendo estas posteriormente integradas. Cada actividade pode utilizar um conjunto de produtos assistentes: *frameworks*, padrões de desenho e mecanismos abstractos. No secção seguinte será apresentada uma *framework* que assiste o desenho do aspecto de concorrência nos pontos 5. e 7.

3.1.1.1 Framework para geração e controlo de concorrência

O trabalho apresentado na terceira parte do relatório recorreu à utilização da *framework* de concorrência desenvolvida no âmbito do projecto DASCo, que se encontra documentada em [Gil 96], [Silva 96b] e [Silva 98] e foi desenvolvida no grupo de Engenharia de *Software* do INESC pelos Eng.os Luís Gil, João Martins e António Rito Silva. A realização foi feita em C++ sobre o ambiente ACE [Schmidt 94].

O sistema suporta diversos padrões de desenho orientados a objectos [Gamma 95] e define uma arquitectura em três níveis : Aspecto, Composição e Aplicação.

No **nível de aspecto** são implementados padrões independentes e abstractos que oferecem soluções para problemas simples. Estas soluções são encapsuladas de forma a permitir a sua posterior composição. Os padrões implementados são:

- **Sincronização de objecto**. Permite suportar diversas políticas de controlo de concorrência sobre um recurso partilhado.
- **Objecto activo**. Separa a execução de um método da sua invocação. Simplifica a sincronização no acesso a objectos partilhados. Define uma política própria de atendimento.
- **Recuperação**. Distingue a política de recuperação de um objecto do algoritmo da aplicação.
- **Geração de concorrência**. Define políticas de geração de concorrência, separando-as do controlo.

O **nível de composição** é formado por componentes responsáveis pela integração dos padrões e por tornar as suas interdependências coerentes. Estes componentes apresentam uma interface simples de modo a que o programador as integre nas suas aplicações de forma quase transparente. Estão disponíveis as composições de padrões compatíveis e que se apresentam coerentes. A composição é feita recorrendo ao mecanismo de herança múltipla.

Em cada aplicação são feitas as adaptações necessárias a partir dos componentes de integração disponibilizados. O **nível aplicação** apresenta uma implementação específica para cada aplicação pretendida, definindo as características dos objectos envolvidos.

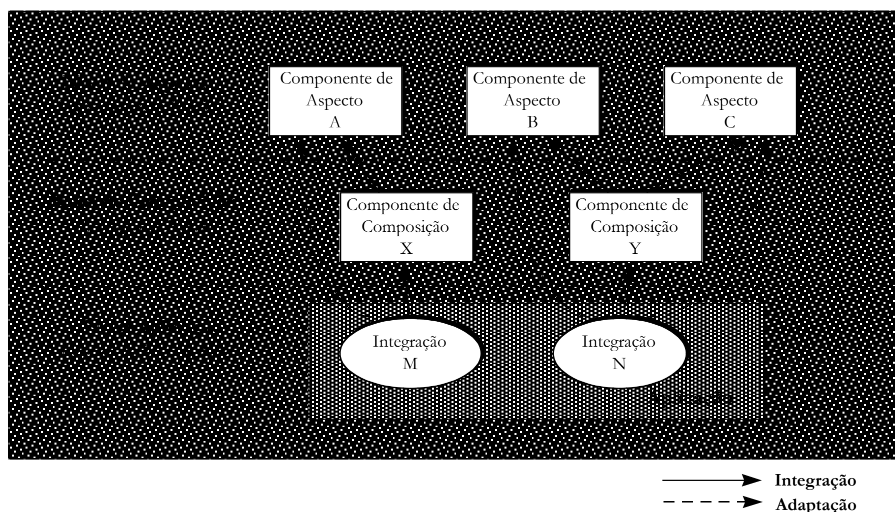


Figura 3 - Arquitectura da *framework*.

A Figura 3 representa a arquitectura descrita. Nela podem observar-se as relações de adaptação e integração. As composições integram os padrões e cada aplicação específica é responsável por adaptar cada integração em pontos específicos dos padrões.

De seguida apresentam-se alguns fragmentos dos dois níveis implementados pela *framework* que ilustram os seus conceitos e estrutura. Finalmente, apresenta-se a sua utilidade no domínio das aplicações de trabalho cooperativo.

3.1.1.1.1 Padrão de sincronização de objecto

O padrão de sincronização de objecto suporta uma abstracção sobre diversas políticas de sincronização, separando-a da geração da concorrência e da funcionalidade do objecto. Desta forma é possível criar políticas independentes (quer de sincronização, quer de geração de concorrência) para aplicações diferenciadas.

Em seguida apresenta-se a descrição do padrão de sincronização de objecto [Silva 96a][Silva 97c], do qual a *framework* apresenta uma implementação particular. A Figura 4 ilustra a comunidade de objectos envolvidos no padrão de sincronização de objecto e as relações existentes entre os participantes. Os diagramas de modelação de objectos são apresentados recorrendo à *Unified Modeling Language* (UML) [Rational 97]. No Apêndice A pode encontrar-se uma introdução aos conceitos da linguagem que foram utilizados neste documento.

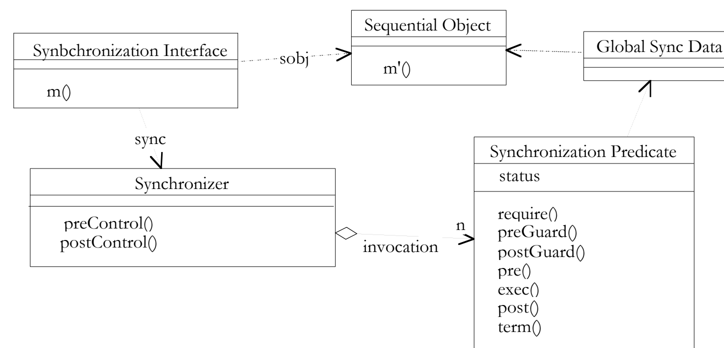


Figura 4 - Padrão de sincronização de objecto

Os participantes são:

- ***Sequential Object***. Contém o código sequencial e os dados a partilhar pelas actividades de concorrência.
- ***Synchronization Interface***. Sincroniza as invocações ao *Sequential Object*. Utiliza os serviços disponibilizados pelo *Synchronizer*.
- ***Synchronizer***. Controla a sincronização das invocações decidindo para cada invocação, se ela pode continuar ou deve ser bloqueada.
- ***Synchronization Predicate***. Contém a semântica da invocação, o que permite determinar se uma condição deve ou não continuar a sua execução. Ex. No algoritmo de leitores/escritores [Marques 90] existem invocações com predicados de leitura e outras com predicados de escrita.
- ***Global Sync Data***. Gere os dados globais de sincronização para um objecto. Ex. Para implementar o algoritmo de leitores/escritores mantém-se o número de leitores, escritores, etc. O controlo de acesso baseia-se nestes dados.

É ainda importante identificar a separação que existe entre as formas de adaptação aplicáveis:

- **Específicas do objecto**. Centram-se na semântica dos objectos, na sua estrutura e operações. A adaptação é feita em subclasses de *Synchronization Predicate* e *Global Sync Data*.

- **Independentes do objecto**. Dependem da semântica da aplicação e implementam-se por especialização da classe *Synchronizer*. As novas classes deverão ser admitidas na interface do objecto (*Synchronization Interface*).

A implementação do padrão na *framework* recorre à parametrização das classes para permitir a introdução de diferentes tipos de invocações sobre os objectos. Cada tipo de invocação é implementado por uma classe específica. A cada invocação é atribuído um *Synchronization Predicate*.

3.1.1.1.2 Padrão de recuperação de objecto

Tal como o padrão de sincronização de objecto, o padrão de recuperação de objecto [Silva 97a], suporta uma abstracção sobre diversas políticas de recuperação de objectos, separando a recuperação do código sequencial do objecto.

A Figura 5 apresenta a comunidade de objectos envolvidos no padrão de recuperação de objecto e as relações entre os participantes.

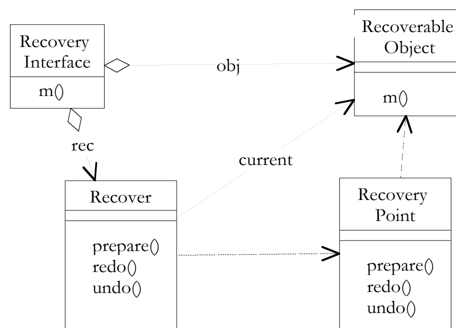


Figura 5 - Padrão de recuperação de objecto

Os participantes são:

- ***Recovery Interface***. Intercepta as invocações dirigidas e executadas sobre o *Recoverable Object*. Estas operações tornam-se recuperáveis;
- ***Recover***. Define a parte da política de recuperação que é independente do objecto. Políticas particulares são definidas em subclasses, tornando-se reutilizáveis;
- ***Recovery Point***. Define a semântica de recuperação específica do objecto. As suas operações podem utilizar o estado interno do *Recoverable Object*;
- ***Recoverable Object***. Encapsula os dados que podem ser recuperados. Podem existir várias instâncias para suportar a recuperação.

Políticas de adaptação aplicáveis:

- **Independentes do objecto**. Podem ser utilizados dois algoritmos genéricos:
  - **Alteração no local (*Update-in-place*)**. O *Recoverable Object* corrente reflecte todas as alterações. Surge habitualmente associado a controlo pessimista da concorrência;
  - **Alteração deferida (*Deferred-update*)**. As alterações não são feitas sobre o *Recoverable Object* corrente. Surge habitualmente associado a controlo optimista da concorrência.
- **Específicas do objecto**. Suportadas pelos *Recovery Point*'s. Existem duas soluções genéricas:
  - **Cópia**. A política usa cópias do estado do objecto para recuperar a um estado anterior;
  - **Compensação**. São usadas operações inversas para repor o estado de um objecto.

Tal como foi visto no padrão de sincronização de objecto, a implementação do padrão de recuperação na *framework* recorre à parametrização das classes para permitir a introdução de diferentes tipos de invocações sobre os objectos.

Uma descrição dos restantes padrões implementados na *framework* pode ser encontrada em [Silva 97b] e [Gil 96].

3.1.1.1.3 Exemplo de composição: Composição passiva síncrona recuperável

Apresenta-se agora um exemplo de composição de padrões, implementado pela *framework*. Seleccionou-se a composição recuperável síncrona, dada a sua simplicidade e a sua importância no contexto do trabalho. Esta composição integra os padrões de sincronização e recuperação de objecto.

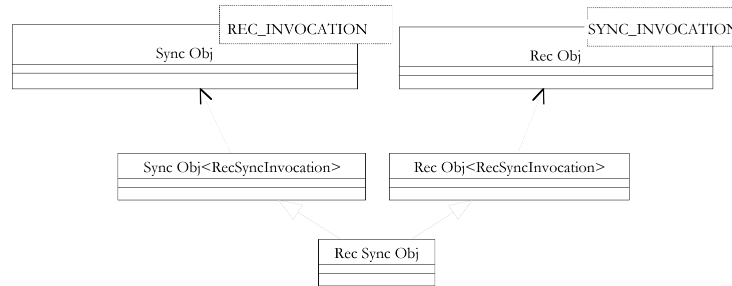


Figura 6 - Composição passiva síncrona recuperável

A integração é feita recorrendo ao mecanismo de herança múltipla. Como se pode observar na Figura 6 a manutenção da coerência e a construção de uma interface mínima para a composição é feita por parametrização de classes. A outro nível é feita por introdução de constrangimentos no código da nova classe, nomeadamente no seu construtor (Ex. ambas as implementações dos padrões mantêm uma fila de invocações. Para manter a coerência, cada nova invocação deverá ser introduzida nas filas de cada um dos padrões.)

3.1.1.1.4 A framework no suporte de aplicações de trabalho cooperativo

Um estudo cuidado da *framework*, revela que esta apresenta características que a destacam como uma solução apropriada para o desenvolvimento de componentes de controlo da concorrência em sistemas de trabalho cooperativo interactivos convergentes. Este facto demonstra-se recorrendo a um exemplo simples e genérico.

Imagine-se uma aplicação em modo *WYSIWIS* onde se encontram diversos ícones sobre um espaço público, onde cada utilizador pode movimentá-los, organizando a sua disposição como pretender. É óbvio que se apresenta um problema concreto de coerência de dados se dois ou mais intervenientes tentarem movimentar um mesmo ícone simultaneamente. A situação é ilustrada de forma simples na Figura 7.

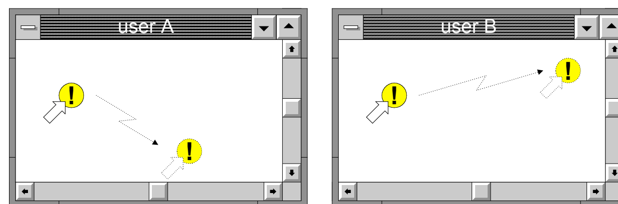


Figura 7 - Situação de conflito

Vejamos pois, como se poderá utilizar a *framework* para implementar soluções para o problema apresentado.

De forma a realizar um desenvolvimento experimental, sugere-se a arquitectura representada na Figura 8.

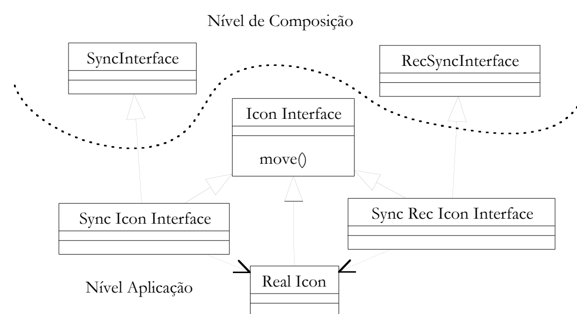


Figura 8 - Arquitectura de dados para desenvolvimento experimental de soluções para controlo da concorrência

A arquitectura apresentada serve de modelo para o desenvolvimento experimental, dados os seus graus de liberdade que poderão ser explorados passo a passo:

- **Introdução do controlo da concorrência**. A arquitectura baseia-se no padrão *proxy* [Gamma 95] permitindo testar a interface para o objecto sequencial, com ou sem controlo da concorrência.
- **Escolher a integração de padrões adequada**. O mecanismo de herança múltipla implementado permite substituir facilmente o *proxy* actual por outro que dependa de uma outra integração do nível de composição.
- **Integrar os padrões de forma progressiva**. O mesmo mecanismo descrito no ponto anterior permite integrar os padrões de faseadamente. Ex. Utilizar inicialmente o mecanismo de sincronização passiva com controlo pessimista, integrando posteriormente os mecanismos de recuperação para resolver situações criadas com a introdução de controlo optimista.
- **Seleccionar políticas específicas à aplicação, independentes dos objectos**. Uma vez seleccionada uma integração e implementado um *proxy* nele baseado, é possível adaptar os padrões seleccionando políticas gerais à aplicação e independentes do objecto. Ex. Escolher entre as políticas optimista e pessimista utilizando diferentes *Synchronizer*'s.
- **Seleccionar políticas específicas aos objectos, independentes da aplicação**. Da mesma forma que foi descrito no ponto anterior, adapta-se a semântica de acesso aos objectos. Ex. Implementação do algoritmo de leitores/escritores em *Global Sync Data* e *Syncronization Predicate*, permitindo a sua evolução, realizando alterações localizadas e independentes da aplicação e da parte sequencial do objecto.

Na secção 2.2 foram apresentados requisitos gerais para controlo da concorrência de sistemas de trabalho cooperativo em tempo-igual. Neste momento torna-se possível estabelecer algumas considerações gerais para o desenho das soluções apontadas, recorrendo ao suporte da *framework*:

- A introdução de mecanismos optimistas de controlo de concorrência é directamente suportada pelo padrão de sincronização de objecto;
- Os requisitos de coerência envolvem na generalidade a manutenção de conjuntos de estados estruturados por uma ordem parcial definida no tempo. Esta estrutura é suportada pelo padrão de recuperação;
- A semântica das interacções pode ser implementada separadamente da semântica dos dados, devendo ainda ser analisada nos diversos aspectos envolvidos em cada padrão: acesso a um objecto, recuperação de estado, autonomia de objecto e acesso a conjuntos de objectos. É suportada pelos diferentes membros das comunidades de objectos que implementam os padrões;
- Novas componentes de composição e novas implementações dos aspectos devem responder aos requisitos de não transparência levantados pela monitorização. Alguns métodos e dados que

implementam as políticas de sincronização e recuperação devem ser tornados públicos e utilizados directamente através da interface utilizador.

3.1.2 Estruturação da interacção

É apresentado um trabalho que apresenta uma proposta de metodologia de cooperação e uma plataforma de desenvolvimento de aplicações.

3.1.2.1 EGRET

O *EGRET* [Johnson 94] é uma *framework* que disponibiliza mecanismos para construção de aplicações que aplicam o paradigma da colaboração exploratória.

O modelo de dados do *EGRET* inclui os seguintes tipos de objectos:

- **Nós**. Uma **Instância de Nó** é um repositório básico de informação, composto por um conjunto de **Campos**. Cada instância de Nó é associada a um **Esquema de Nó**. Um Esquema é uma especificação por defeito do conjunto de Campos que compõem cada Instância de Nó;
- **Campos**. Segundo uma orientação hipertexto, os campos podem conter arbitrariamente texto e **Ligações** para outros Nós. Cada campo mantém também uma associação com um **Esquema de Campo**;
- **Ligações**. São relações direccionadas de Campos para Nós. Existem também **Esquemas de Ligação** que desempenham um papel idêntico ao descrito para os Esquemas anteriores;
- **Camada**. Permitem criar partições no espaço de dados, contendo eventualmente dados replicados incoerentes entre partições. Conceptualmente, assemelham-se a espaços de nomes.

O modelo de processo de colaboração desenvolve-se em três fases:

1. **Fase Consensual**. A interacção inicia-se a partir de uma estrutura consensual, ou seja de um conjunto de Nós, Ligações e respectivos Esquemas, coerentes para os diversos membros do grupo;
2. **Fase de Exploração**. Durante esta fase os indivíduos podem alterar os Nós, Ligações e Esquemas livremente;
3. **Fase de Consolidação**. A *framework* providencia serviços capazes de identificar as diferenças entre as versões de Nós e Esquemas. Uma vez identificadas, procede-se a uma das seguintes actividades:
    - As Instâncias são modificadas de acordo com os Esquemas;
    - Os Esquemas são modificados de acordo com as Instâncias;
    - Cria-se uma nova Camada, para onde são migradas as entidades relevantes.

As duas primeiras actividades ocorrem quando as diferenças são pequenas e localizadas. Quando forem necessárias reestruturações mais complexas recorre-se à última hipótese. A alteração e evolução de Esquemas e Camadas pode ser conduzida pelo utilizador ou por agentes autónomos. No final desta fase atinge-se um estado em que se pode iniciar uma nova Fase Consensual.

A plataforma desenvolvida funciona numa arquitectura cliente-servidor. Os processos cliente podem ser agentes de *software* ou agentes humanos.

3.1.3 Interface utilizador

Apresenta-se uma ferramenta que permite criar interfaces utilizador sobre estruturas de dados baseadas em grafos, o EdGar [Paulo 91]. A ferramenta foi construída pelo Eng.º Vasco Paulo no Grupo de Técnicas de Interacção e Multimédia do INESC.

3.1.3.1 EdGar

Embora não tenha sido construído com o propósito de desenvolver interfaces utilizador, o EdGar apresenta-se especialmente vocacionado para tal, uma vez que introduz mecanismos de construção da interface utilizador, sobre estruturas de grafos, podendo estes estruturar os dados de modo hipertexto. Por outro lado, fornece mecanismos para manipulação destes através da interface utilizador.

Genericamente, a ferramenta permite a criação e edição dos nós e ligações de grafos sob várias representações visuais. Para tal, é fornecida uma biblioteca C++ inicialmente concebida para as plataformas *X-Windows/Unix*, tendo sido posteriormente portada para *Windows*.

A características principais que pesaram na escolha do EdGar como suporte da interface são:

- Criação e edição de estruturas de tipo grafo, fornecendo a possibilidade de implementar organizações do tipo hipertexto;
- Manipulação interactiva das estruturas criadas;
- Permite escolher entre uma variedade de representações gráficas.

A biblioteca é composta de cinco módulos fundamentais:

- **List**: Implementa classes para manipulação de vários tipos de listas.
- **Util**: Define tipos de dados abstractos de utilização comum. Ex. cadeias de caracteres, geradores de identificadores, comandos, grafos, nós, ligações, etc.
- **Graphics**: Nível intermédio que implementa as funcionalidades de interface utilizador, de forma independente da plataforma e dos grafos. Exemplos: rectângulos, *bitmaps*, texto, etc.
- **WGraph**: Especialização de *Graphics* para a plataforma *Windows*.
- **XGraph**: Especialização de *Graphics* para *X-Windows/Unix*.
- **View**: Definição de classes para visualização e edição de grafos, definindo o código que permite tratar da estruturação da interacção com o utilizador, ligando-o aos eventos dos sistemas operativos e dos sistemas de janelas.

Esta ferramenta foi utilizada na construção das aplicações apresentadas em 3.2.1 e do protótipo apresentado no Capítulo 7.

3.2 Aplicações

O objectivo desta secção é apresentar aplicações existentes que apresentam preocupações semelhantes às deste projecto. São apresentadas duas famílias de aplicações. A primeira (*NGTool* e *NGMeeting*) serviu como primeira aproximação às técnicas de interacção aqui apresentadas, embora a sua realização tenha focado especialmente as técnicas facilitadoras de reuniões. A segunda (*GroupSystems*) introduz um produto comercial com funcionalidade idêntica. A decisão de estudar este sistema pretende apenas analisar superficialmente o estado da tecnologia nesta área de forma a estabelece um paralelo com os resultados finais deste trabalho.

3.2.1 NGTool e NGMeeting

A aplicação *NGMeeting* [Ho 96] realizada pelas Eng.as Tânia Ho e Isabel Soares no Grupo de Técnicas de Interacção e Multimédia do INESC, surge como extensão à ferramenta *NGTool* [Antunes 95c] realizada pelo Prof. Pedro Antunes.

A área de investigação sob a qual foram realizados ambos os projectos, centrou-se no desenho de sistemas de suporte à interacção e tomada de decisão em grupo, baseados em técnicas de comportamento social. Este tema envolve bastantes considerações que estão fora do âmbito do trabalho e que podem ser

encontradas em [Antunes 95b] e [Antunes 96b]. De forma generalista, classificam-se como aplicações de suporte à realização de reuniões distribuídas.

O estudo destes trabalhos é feito em duas vertentes:

1. Estudo das características de suporte à interacção, enquadrando-os nos conceitos apresentados no Capítulo 2.
2. Apresentação dos detalhes técnicos da aplicação *NGMeeting*. Esta apresentação serve para enquadrar a concepção do protótipo apresentado no Capítulo 7 e que teve como base esta aplicação.

3.2.1.1 Desenho das aplicações

Como foi dito, ambas as aplicações estruturam as sessões de trabalho segundo técnicas facilitadoras de reuniões, mais concretamente:

- ***Nominal Group Technique***. Suportada por ambas as aplicações, estrutura as reuniões em cinco passos:
  1. Introdução à reunião por parte de um moderador;
  2. Geração individual de ideias por escrito;
  3. Registo circular de ideias;
  4. Clarificação e discussão em grupo de ideias apresentadas;
  5. Votação individual de ideias e discussão dos resultados.
- ***Brainstorming***. Suportada pela aplicação *NGMeeting*, permite a geração livre de ideias, seguida de uma discussão e votação.
- **Votação**. O *NGMeeting* suporta um vasto conjunto de técnicas para suporte à tomada de decisão em grupo.

As técnicas descritas apresentam um primeiro grau de estruturação da interacção em grupo. No entanto, as técnicas de suporte directo que são relevantes no contexto deste trabalho são apresentadas de seguida.

3.2.1.1.1 Estruturação da interacção

A construção da aplicação seguiu o modelo apresentado em [Antunes 96a] e [Antunes 96b] sendo considerados dois tipos de elementos estruturantes que podem ser públicos ou privados:

- **Registos**. São repositórios de informação concreta e durável que permitem a manipulação e partilha de dados.
- **Assistentes**. Os assistentes são associados aos dados partilhados e servem para monitorizar os processos de grupo. Desempenham um papel fundamental na monitorização do controlo da concorrência.

Sobre eles é considerada uma operação estruturadora (podendo outras ser definidas ao nível da aplicação):

- **Conexão**. Estende o número de operações suportadas pela interface utilizador permitindo a associação de dois objectos através de uma única acção. Estas associações têm como objectivo a estruturação dos dados partilhados (conexões entre registos) e execução de acções estruturadoras da interacção (conexões entre assistentes e registos).

3.2.1.1.2 Interface utilizador

Os elementos estruturantes são materializados ao nível da interface utilizador por ícones. As associações criadas pelas operações de Conexão são representadas por linhas. São definidos dois espaços distintos: o espaço público e o espaço privado, definindo estes a propriedade de visibilidade de cada um dos objectos.

Os dados mantidos nos repositórios dos registos podem ser manipulados junto da sua representação gráfica recorrendo a janelas de diálogo.

A Figura 9 apresenta uma reprodução da interface utilizador da *NGTool*. A interface da aplicação *NGMeeting* é idêntica.

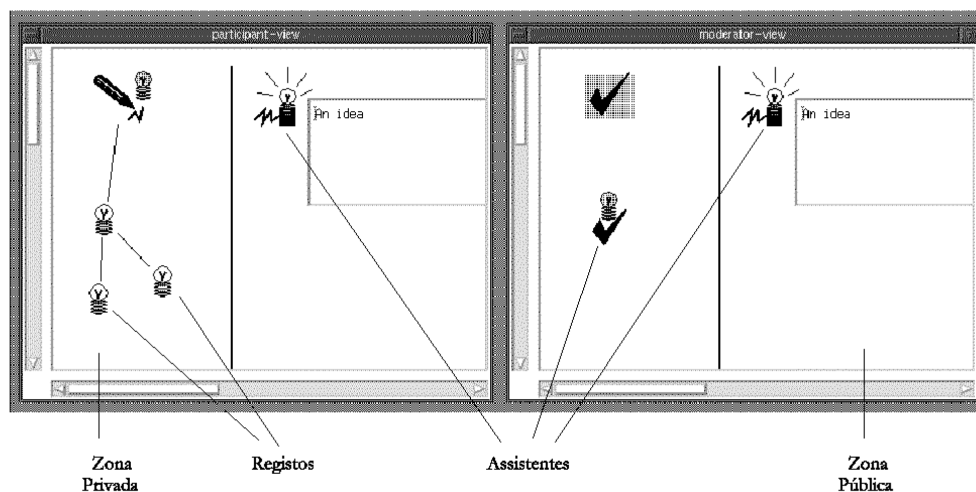


Figura 9 - Interface gráfica do *NGTool*

A imagem ilustra a representação dos elementos estruturadores. As ligações entre os registos foram conseguidas por estabelecimento de conexões. Observa-se a divisão existente entre espaços público e privado. O modo de interacção com os registos é do tipo *WYGIWIG* sendo relaxada a coerência nos aspectos de monitorização.

3.2.1.1.3 Partilha de dados

As ferramentas *NGTool* e *NGMeeting* apresentam uma arquitectura distribuída com replicação de dados. Sobre ela é feito um controlo pessimista da concorrência, incorporando um mecanismo de controlo de acesso baseado em privilégios para os criadores dos registos.

Figura 10 - Controlo da concorrência (O utilizador da esquerda requer o trinco que é detido pelo utilizador da direita)

Ao nível da interface é feita a monitorização deste controlo utilizando um conjunto de assistentes baseados na metáfora de uma mão. A título de exemplo observe-se a Figura 10.

3.2.1.2 Realização

Apresentam-se agora os detalhes de realização da ferramenta *NGMeeting* fundamentais à compreensão trabalho apresentado no Capítulo 7.

A ferramenta foi construída para a plataforma *Windows* e é composta pelos módulos representados na Figura 11.

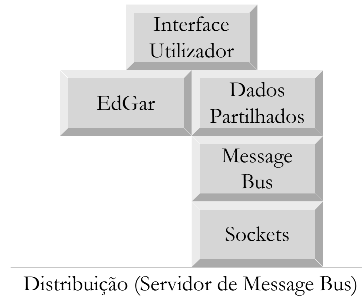


Figura 11 - Módulos da aplicação *NGMeeting*

As funcionalidades mais básicas da interface utilizador têm por base as *Microsoft Foundation Classes* e as mais estruturadas, são construídas sobre o EdGar.

3.2.1.2.1 MBus

O módulo de dados partilhados realiza a estrutura das reuniões, incluindo o processamento e emissão de eventos por distribuição. A difusão das mensagens recorre à utilização do *MBus*. [Kaplan 92]. Este sistema permite a comunicação entre múltiplas aplicações, através da difusão de mensagens selectivas. A difusão é simulada por um servidor central de *MBus* que é responsável por difundir as mensagens recebidas pela(s) classe(s) de utilizadores indicada(s) na mensagem. As mensagens assumem um formato com a seguinte estrutura:

(*tag domains mensagem*)

O campo *tag* indica a classe de mensagem a enviar. Os *domains* indicam a classe de utilizadores para quem a mensagem é dirigida. O último campo permite enviar um conjunto de dados no formato pretendido pelas aplicações.

O sistema *MBus* funciona sobre *sockets TCP-IP*, fornecendo uma biblioteca de funções para a sua utilização a um nível de abstracção elevado.

Mecanismos mais específicos de cada uma das ferramentas e módulos enunciados serão pontualmente referidos na terceira parte do relatório.

3.2.2 GroupSystems

O *GroupSystems* é também um sistema de suporte à tomada de decisão em grupo. Para tal, providencia módulos que suportam diferentes processos de geração de ideias:

- ***Categorizer***. Permite gerar listas de ideias e comentários associados. Posteriormente são definidas categorias que permitem organizar as ideias e comentários gerados;
- ***Electronic Brainstorming***. Apresenta um mecanismo simples para discussão de um determinado assunto, através da apresentação de ideias e posterior comentário;
- ***Group Outliner***. Permite a um grupo criar e comentar uma organização hierárquica de ideias. A hierarquia pode ser apresentada através de diversas representações simples;
- ***Topic Commenter***. Oferece a possibilidade de comentar um conjunto de tópicos. É mais estruturado que o *Electronic Brainstorming*, mas menos do que o *Group Outliner*;
- ***Vote***. Implementa um conjunto de técnicas de votação que auxiliam a tomada de decisão sobre um determinado tema.

O *GroupSystems* é uma aplicação comercial, pelo que não estão disponíveis detalhes de implementação ao nível da partilha de dados. No entanto, é possível ao nível da interface depreender os mecanismos de controlo da concorrência que são utilizados.

O controlo é pessimista e realizado sobre um modo de interface composto pelos modos *WISIWIMS* e *WYGIWIG*. A Figura 12 apresenta um exemplo de utilização no módulo de *Electronic Brainstorming*.

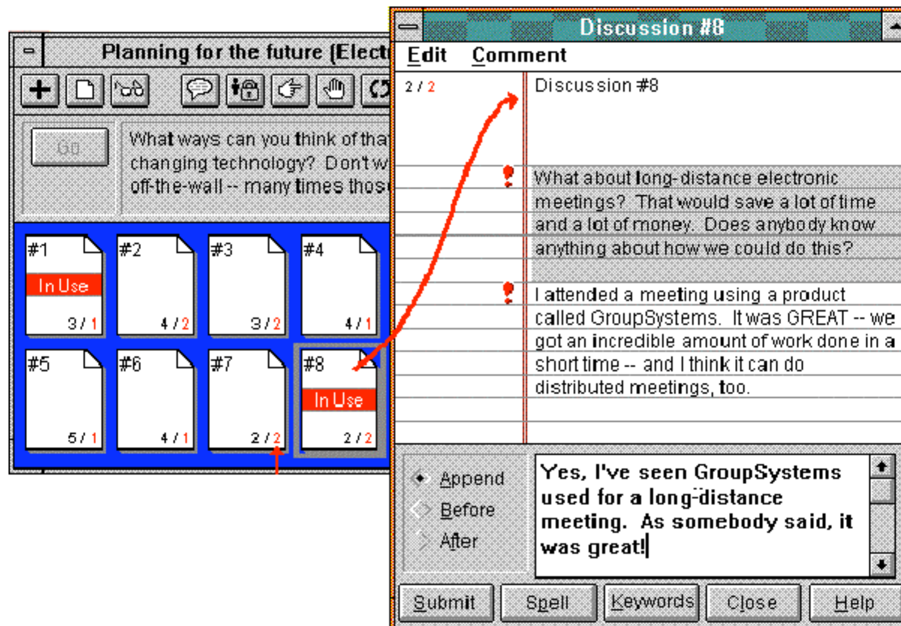


Figura 12 - Exemplo de utilização do *GroupSystems*

Como se pode observar, as ideias estão organizadas em grupos de discussão, recorrendo à metáfora da folha. O controlo pessimista da concorrência é monitorizado pelas faixas vermelhas colocadas nas folhas. As alterações são monitorizadas pelos pontos de exclamação e pelos números no canto inferior direito das folhas. A edição das ideias é feita num espaço privado (zona inferior da janela de discussão). Só no final da edição é feita a submissão da totalidade da ideia.

O mecanismo apresentado é de implementação simples mas introduz interacções pouco consonantes com as regras de comportamento social. A obtenção do trinco gera tempos de espera pouco agradáveis. A impossibilidade de editar livremente qualquer ideia reduz o paralelismo e a livre circulação de ideias.

Parte II

MODELO DE ESTRUTURAÇÃO DA INTERACÇÃO

Capítulo 4

SUPORTE À INTERACÇÃO

É feita a descrição de um modelo que permite suportar e controlar a interacção em grupo para aplicações de trabalho cooperativo em tempo-igual/local-diferente. O modelo apresenta conceitos dedicados aos três aspectos diferentes do suporte de interacção: partilha de dados, estruturação da interacção e interface utilizador. Neste Capítulo apresentam-se os mecanismos de suporte.

4.1 Introdução

O modelo apresentado neste Capítulo e no seguinte, diz respeito à componente de estruturação da interacção e sua influência na interface utilizador e na partilha de dados. Consideram-se dois aspectos fundamentais:

- **Suporte**. Descreve os mecanismos que permitem efectivar a cooperação;
- **Controlo**. Estuda as técnicas que permitem reduzir os conflitos gerados sobre os mecanismos de suporte.

Surge como uma reformulação e extensão do modelo apresentado em [Antunes 96a] e [Antunes 96b] e já abordado em 3.2.1.1.1. Faz o suporte à estruturação da interacção de forma independente da interface utilizador e da partilha dos dados, influenciando no entanto o desenho dessas componentes:

- Os **dados partilhados** são estruturados de forma hierárquica. Existem objectos públicos e privados, podendo esta propriedade ser alterada dinamicamente;
- A **interface utilizador** é também baseada na existência de objectos públicos e privados sendo os públicos apresentados segundo o modelo *WYGIWIG* dadas as condicionantes de tempo na distribuição dos eventos. O modo *WYSIWIMS* pode ser ou não utilizado.

Os mecanismos de controlo da concorrência e de relaxação de coerência serão estudados no Capítulo seguinte, sendo independentes do suporte aqui apresentado.

4.2 Objectos

Introduzem-se quatro tipos de objectos de interacção diferentes: **Contentor**, **Conteúdo**, **Conexão** e **Monitor**. Os dois primeiros dizem respeito à partilha dos dados, o terceiro à estruturação da interacção, e o quarto à monitorização das actividades dos membros do grupo.

4.2.1 Contentores e Conteúdos

Os **Contentores** representam e estruturam toda a informação relevante às aplicações. São objectos funcionais que permitem manipular as suas componentes:

- **Representação**. Objecto gráfico que identifica o Contentor. Componente de interface utilizador;
- **Conteúdo**. Componente de partilha de dados. Armazena a informação associada ao Contentor. Os Conteúdos apresentam uma propriedade de **estruturação**:
  - Um Conteúdo **simples** é um conjunto de dados de tipo a definir ao nível da aplicação;
  - Um Conteúdo **composto** contém um conjunto de Contentores.

Esta propriedade, ao contrário de outras que veremos adiante, é estática e pré-definida para cada tipo de Contentor ao nível da aplicação. Os contentores com Conteúdo composto definem um unidade de estruturação de informação. De modo a simplificar o modelo apresentado, consideram-se estruturas hierárquicas. Isto é, cada Contentor pode pertencer a apenas um Conteúdo composto.

Cada Conteúdo tem um componente próprio:

- **Apresentação**. Objecto gráfico que medeia a manipulação dos dados da aplicação por parte do utilizador.

4.2.1.1 *Propriedades dinâmicas*

Os Contentores e os Conteúdos apresentam as seguintes propriedades dinâmicas:

- **Visibilidade**. Dizem-se **públicos** ou **privados**, consoante são ou não visíveis e manipuláveis por todos os membros do grupo. A propriedade de visibilidade de um Contentor define também a do seu Conteúdo;
- **Manipulação**. Podem classificar-se como **duráveis** ou **transitórios**.
  - Os **Contentores** quando duráveis são inamovíveis entre Contentores compostos e podem representar geradores de novos Contentores e/ou repositórios de unidades de dados. Os transitórios são móveis e permitem estruturar ou eliminar unidades de dados;
  - Os **Conteúdos** quando duráveis são insubstituíveis e inamovíveis. Representam unidades de dados e permitem duplicá-los e compô-los. Quando transitórios representam unidades de dados voláteis, permitindo mover e eliminar dados ao nível da aplicação;

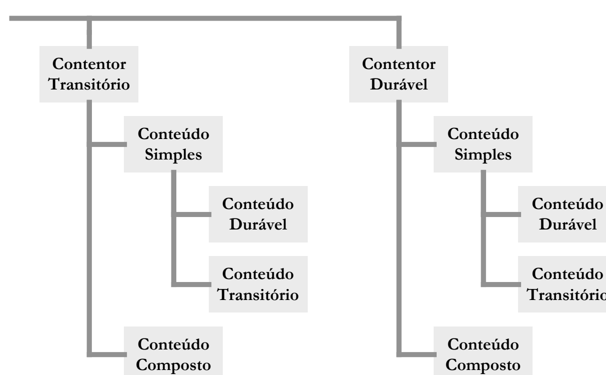


Figura 13 - Hierarquia de propriedades de cada Contentor

A Figura 13 resume as propriedades admissíveis para cada par Contentor/Conteúdo como unidade de estruturação de informação. Este conjunto de propriedades define o seu comportamento perante as operações de estruturação da interacção.

4.2.2 *Conexões*

Modelam a estruturação da interacção sobre pares de Contentores. Parte da semântica de uma Conexão é definida à custa das propriedades de visibilidade e manipulação dos Contentores e Conteúdos envolvidos,

sendo possível identificar padrões de interacção genéricos aplicáveis a qualquer aplicação. A semântica pode ainda ser enriquecida ao nível da aplicação desde que não viole os padrões apresentados.

Uma Conexão tem três componentes:

- **Origem**. Contentor a partir do qual é iniciada a Conexão;
- **Destino**. Contentor ou espaço livre para o qual é direccionada a Conexão;
- **Transferência**. Um Contentor (novo ou já existente) que pode ser utilizado pelo Contentor destino. Representa as trocas de informação resultantes da interacção.

As Conexões são feitas em duas fases:

- **Iniciação**: Cria-se a Transferência em função da Origem e altera-se a Origem (se necessário);
- **Finalização**: Altera-se o Destino em função da Transferência (se necessário).

As figuras seguintes apresentam os padrões de interacção identificáveis a partir das propriedades de manipulação e composição dos Contentores e Conteúdos envolvidos. Referem-se apenas as propriedades relevantes nas interacções.

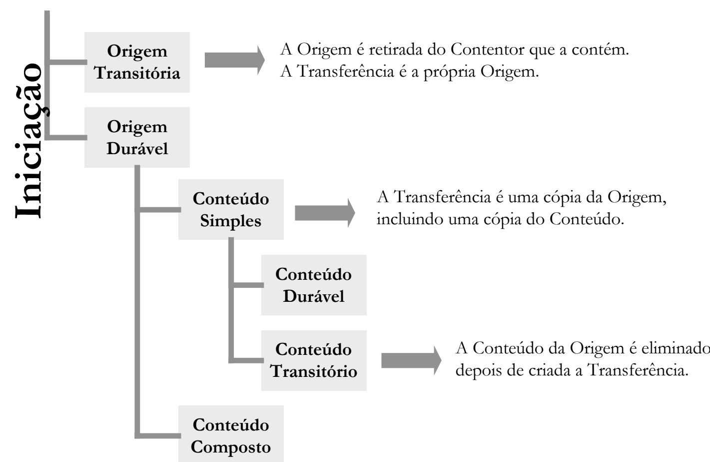


Figura 14 - Afectação dos objectos envolvidos na fase de Iniciação

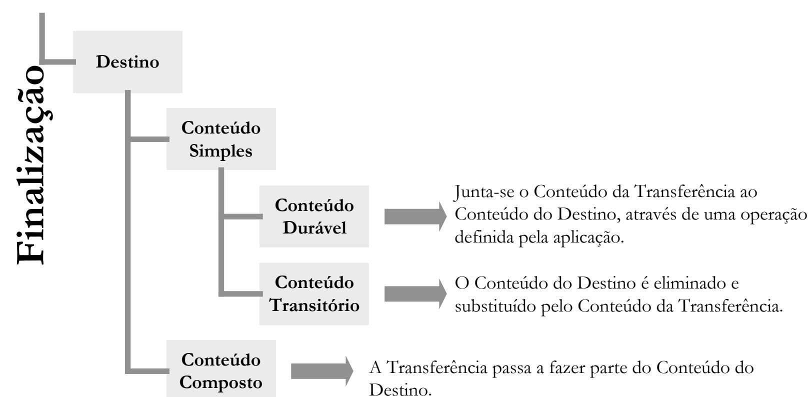


Figura 15 - Afectação dos objectos envolvidos na fase de Finalização

Cada uma das entradas das Figura 14 e 15 descreve uma componente ou operação sobre os objectos de interacção envolvidos. A descrição completa da funcionalidade de cada padrão de interacção obtém-se por composição dos aspectos apontados. Os aspectos não especificados podem ser definidos ao nível das aplicações.

Considerem-se pois alguns exemplos simples, ilustrativos das funcionalidades do modelo apresentado. Considerações de interface utilizador serão tomadas posteriormente.

1. Estabelecimento de uma conexão entre um Contentor durável com Conteúdo simples e um Contentor Composto:
   - **Execução**. É criada a Transferência como sendo uma cópia da Origem com uma cópia do Conteúdo da Origem. O Conteúdo do Destino passa a conter o novo Contentor;
   - **Funcionalidade**. Duplicação de um conjunto de dados para uma unidade de informação distinta.
2. Estabelecimento de uma conexão entre um Contentor transitório composto e outro Contentor composto:
   - **Execução**. A Origem é eliminada do Contentor que a acolhe. A Transferência é a própria origem. O Conteúdo do Destino passa a conter a Transferência;
   - **Funcionalidade**. Reorganização da estrutura de dados mantendo-os inalterados.
3. Estabelecimento de uma conexão entre um Contentor simples transitório e outro Contentor simples:
   - **Execução**. A Transferência é a própria origem. O Conteúdo do Destino passa a conter o Conteúdo da Origem. O Conteúdo da Origem é eliminado;
   - **Funcionalidade**. Migração de uma unidade de dados entre unidades de estruturação de dados.

No Apêndice B apresenta-se um conjunto completo de exemplos ilustrados com figuras.

4.2.3 Monitores

O Monitores são objectos gráficos que apresentam pistas sobre as actividades que os utilizadores exercem sobre os objectos públicos. Como as actividades dos utilizadores são exercidas através dos objectos anteriormente apresentados, os monitores apresentam-se, ao nível da interface utilizador, junto das representações e apresentações dos Contentores e Conteúdos, respectivamente.

Na ausência de eventos interessantes, um monitor não é visível de forma a não introduzir elementos distractivos na interface.

Exemplos de tipos de monitores, são:

- **Monitor de concorrência**. Fornece indicações aos utilizadores sobre o estado de acesso aos objectos de dados. São apresentados mais em pormenor no Capítulo 5.
- **Monitor de propriedade**. Permite visualizar as propriedades dos objectos, quando estas não podem ser inferidas pela disposição da interface. Adiante, é apresentada uma discussão deste tema.
- **Monitor de elasticidade temporal**. Monitorizam a obtenção da informação partilhada por parte dos restantes elementos do grupo. Um exemplo de utilização é apresentado no Capítulo 7.
- **Monitor de gestão de interface**. A utilização de modelos *WYSIWIMS* permite que os utilizadores mantenham vistas incoerentes sobre os mesmos dados. Desta forma, torna-se necessário monitorizar as alterações às estruturas de dados que não sejam directamente monitorizadas por algum elemento da interface da aplicação. Uma discussão deste tema é apresentada posteriormente neste Capítulo.

Outros tipos de monitores podem ser definidos para cada aplicação. Outros ainda que não se encontram directamente relacionados com este trabalho podem ser encontrados em [Antunes 96a] e [Antunes 96b].

4.2.4 *Resumo*

Uma vez apresentados os objectos interessantes do modelo, importa agora enquadrá-los na discussão apresentada no Capítulo 2. Nomeadamente nas três componentes dos sistemas em estudo: estruturação da interacção, partilha de dados e interface utilizador. Este enquadramento é feito pela matriz da Figura 16.

	Interface Utilizador	Estruturação da Interacção	Partilha de Dados	
			Estruturação	Edição
Contentor	√		√	
Conteúdo	√			√
Conexão		√		
Monitor	√			

Figura 16 - Matriz de componentes dos objectos do modelo

Distingue-se entre estruturação e edição de dados, de modo a realçar a diferença entre Contentores e Conteúdos. Os Contentores estruturam Conteúdos. Os Conteúdos mantêm unidades de informação.

Note-se que a estruturação da interacção através da utilização de Conexões pode induzir nos Contentores operações de estruturação de dados.

4.3 Opções de Desenho

Nesta secção pretende-se introduzir uma discussão sobre as opções de desenho deixadas em aberto pelo modelo. A discussão centra-se nos aspectos de interface utilizador e de estruturação da interacção. O aspecto da partilha de dados é fortemente influenciado pelo modelo de controlo da interacção descrito no próximo Capítulo.

Note-se que as considerações aqui introduzidas são apenas sugestões. O modelo admite qualquer possibilidade que responda aos requisitos até aqui enumerados.

4.3.1 *Interface utilizador*

Focam-se os aspectos de: representação de Contentores, apresentação de Conteúdos, e monitorização.

4.3.1.1 *Representação de Contentores*

A forma adoptada para representação dos Contentores na realização do protótipo apresentado no Capítulo 7, foi a utilização de ícones (eventualmente etiquetados). Muitas aplicações utilizam esta representação. A Figura 17 apresenta algumas hipóteses.

Figura 17 - Representações de Contentores

4.3.1.2 *Apresentação de Conteúdos*

A mediação da alteração das unidades de informação é feita recorrendo à componente de apresentação dos Conteúdos. A apresentação de um Conteúdo deve ser feita junto da representação do Contentor. Sendo que a apresentação dos Conteúdos simples é necessária e substancialmente diferente da apresentação dos Conteúdos compostos.

No caso da aplicação desenvolvida no âmbito do trabalho, os dados a manipular são simples cadeias de caracteres pelo que se adoptaram as janelas de diálogos para apresentar e editar os Conteúdos simples (ver Figura 18).

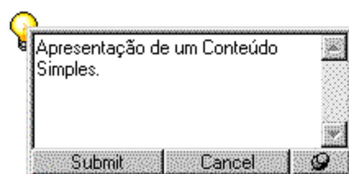


Figura 18 - Apresentação de um Conteúdo simples

Outras aplicações mais complexas podem recorrer a uma variedade de soluções, como por exemplo: lançamento de novas aplicações e utilização de mecanismos de comunicação entre processos, utilização de objectos *OLE* [Brockschmidt 94], *applets Java* [Cornell 96], etc. Apesar destas soluções não terem sido testadas na instância do modelo realizada, acredita-se que elas são exequíveis.

Os Conteúdos compostos têm uma natureza distinta. As suas apresentações devem reflectir a natureza hierárquica da organização da informação. A solução mais habitual é colocar linhas a ligar as representações dos Contentores associados. A ligação deverá permitir distinguir o Contentor que acolhe do Contentor que é acolhido. A Figura 19 apresenta duas hipóteses. A primeira é largamente utilizada em aplicações para a plataforma *Windows*. A segunda foi adoptada para a aplicação exemplo.

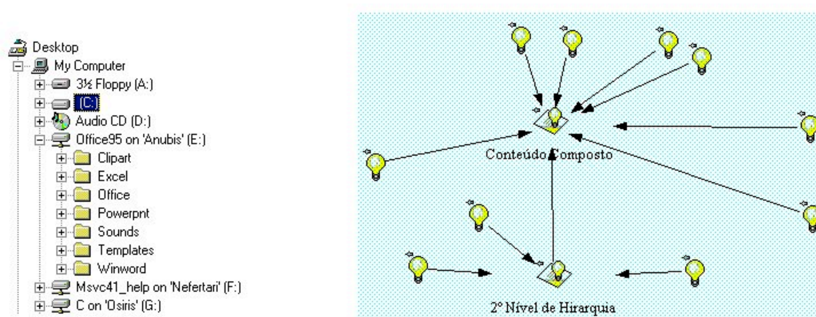


Figura 19 - Apresentações alternativas de Conteúdos compostos

Outras representações são possíveis sem recorrer à utilização de linhas. Por exemplo, a utilização de sistemas de hipertexto em que a vista actual corresponde a um nó da hierarquia e é possível navegar entre vistas. Exemplo: páginas *HTML*.

4.3.1.3 *Monitorização*

As propriedades dinâmicas dos objectos deverão ser monitorizadas, uma vez que a semântica das interacções depende destas. Desta forma o utilizador realiza as acções de forma consciente.

A propriedade de manipulação dos Contentores pode ser monitorizada, recorrendo a um monitor colocado junto da representação do Contentor. Na aplicação exemplo recorreu-se à metáfora do "pionés" (ver Figura 20)

Figura 20 - Monitorização da propriedade de manipulação dos Contentores

A propriedade de manipulação dos Conteúdos simples também deve ser monitorizada, recorrendo à mesma metáfora. O monitor deverá ser visível no interior ou imediações da apresentação do Conteúdo. Na aplicação exemplo recorreu-se à representação da Figura 21.

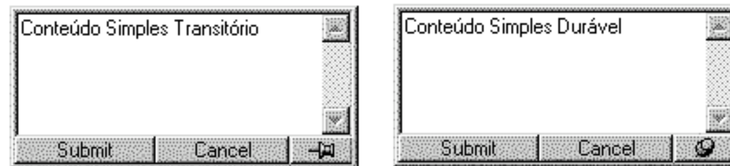


Figura 21 - Monitorização da propriedade de manipulação dos Conteúdos simples

Para monitorizar as alterações de Conteúdos num modo *WYSIWIMS*, importa adornar os Contentores com monitores que indicam que os seus Conteúdos sofreram alterações. Estes monitores aparecem quando essas alterações acontecem. Desaparecem quando o seu Conteúdo é aberto. A Figura 22 apresenta uma hipótese.

Figura 22 - Monitorização de alterações aos Conteúdos num modo *WYSIWIMS*

Na representação dos Contentores de Conteúdo composto devem distinguir-se as situações em que o Conteúdo se encontra aberto ou fechado. A sua monitorização pode ser feita recorrendo a um Monitor ou efectuando variações sobre a representação dos Contentores, ou ambas. As duas situações encontram-se ilustradas na

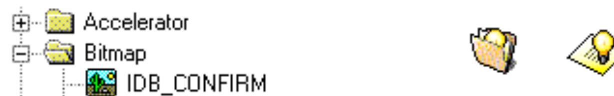


Figura 23 – Monitorização do estado de visualização dos Conteúdos Compostos

A propriedade de visibilidade dos Contentores pode ser identificada de diferentes formas: recorrendo a monitores, dividindo o espaço da janela em zonas públicas e privadas ou realizando variações sobre a representação do Contentor. A utilização de zonas pública e privada foi já apresentada no exemplo da aplicação *NGTool* (Figura 9).

Os monitores de elasticidade temporal permitem tomar consciência da quantidade de membros de um grupo que já visualizaram ou realizaram uma determinada tarefa. Para tal, no caso da aplicação exemplo, recorreu-se a uma barra de progresso, tal como apresentado na Figura 24.

Figura 24 - Monitor de elasticidade temporal

A monitorização da concorrência será discutida no próximo Capítulo.

4.3.2 Estruturação da interacção

Identificou-se a Conexão como objecto estruturador da interacção. No entanto, outras operações existem que afectam a manipulação e estruturação dos dados partilhados.

Como veremos no Capítulo seguinte, o controlo da interacção só é possível se forem identificadas todas estas operações. Interessa pois, enumerá-las e padronizar as formas de realização das operações através dos dispositivos de entrada habituais. De seguida, apresenta-se um estudo sobre o modelo de interacção, identificando as operações possíveis e as acções que as realizam.

- **Estabelecimento de uma Conexão**. Considerando que os Contentores têm um ícone como representação, utiliza-se a acção de *drag'n'drop*, recorrendo ao botão esquerdo do rato. O início do *drag* corresponde à Iniciação e o *drop* à Finalização. Desta forma identificam--se os objectos Origem e Destino. Ao nível da aplicação pode considerar-se que o *drop* é feito, nalguns casos, não directamente sobre o objecto Destino mas sobre uma posição que o indica indirectamente, ou seja, temos acesso a uma "pista" acerca do Destino.
- **Alteração da propriedade de visibilidade**. Esta operação depende da forma como é monitorizada a propriedade. Se for feita recorrendo a um monitor, a alteração poderá ser feita com um *click* do botão esquerdo sobre o monitor. Se o espaço de trabalho for dividido em zonas públicas e privadas, deve considerar-se que cada uma destas zonas é um Contentor de Conteúdo composto. A alteração da propriedade é feita recorrendo a uma Conexão. Note-se que esta propriedade é alterada sempre que é estabelecida uma Conexão e a Origem passa a pertencer a Contentor de Conteúdo composto de propriedade diferente.
- **Alteração da propriedade de manipulação dos Contentores**. Altera-se com um *click* do botão esquerdo sobre o monitor respectivo.
- **Alteração da propriedade de manipulação dos Conteúdos**. Idêntico ao anterior.
- **Abertura da apresentação dos Conteúdos**. É feita com um *double-click* do botão esquerdo sobre a representação do Contentor respectivo.
- **Edição de Conteúdos**. Definido ao nível da aplicação. Ao nível do modelo interessam duas fases: uma fase inicial em que se inicia a edição (esta fase é registada para identificação de possíveis conflitos), e uma fase final em que se submetem os dados para as réplicas do Conteúdo.

A manutenção da coerência entre os espaços públicos é feita por distribuição das acções. O relaxamento visual, consegue-se por não-distribuição ou rejeição de acções. O relaxamento de dados implica a utilização de políticas de controlo que serão objecto do estudo apresentado no próximo Capítulo.

Note-se que ao nível do modelo é possível realizar todas as acções recorrendo a apenas uma tecla do rato. Tal revela a simplicidade e coerência das técnicas de interacção suportadas pelo modelo e, como tal, a facilidade de aprendizagem na utilização de interfaces nele baseadas, bem como, a possibilidade das aplicações poderem estender largamente esse leque de técnicas.

A aplicação apresentada no Capítulo 7 utiliza esta característica, permitindo realizar todas as interacções necessárias (excepto a edição de texto) sem utilizar menus, botões, etc. De modo a simplificar a interacção, é de esperar que toda a semântica das aplicações possa ser captada nos objectos de interacção e respectivas propriedades.

Capítulo 5

CONTROLO DA INTERACÇÃO

Apresenta-se um modelo que permite efectuar o controlo da concorrência sobre os objectos do modelo de suporte definido no Capítulo anterior. O modelo baseia-se nas operações de estruturação da interacção identificadas e na sua semântica. Mais uma vez, apresenta-se um estudo sobre a repercussão do modelo de objectos apresentado na interface utilizador.

5.1 Introdução

A resolução dos problemas de controlo da concorrência em sistemas de trabalho cooperativo interactivos convergentes, apresenta-se como um problema vasto dada a diversidade de técnicas de interacção existentes.

Torna-se no entanto possível, no contexto deste trabalho, apresentar um modelo genérico que assenta sobre o modelo de suporte à interacção apresentado no quarto Capítulo.

Para tal, estuda-se a riqueza de interacção suportada pelo modelo, apresentando-se um conjunto de objectos que solucionam o problema apresentado. Tal como foi feito para o modelo de suporte, introduz-se um conjunto de opções de desenho ao nível da interface utilizador.

As técnicas aqui apresentadas servem apenas para resolver conflitos. A sua utilização deve ser sempre acompanhada das técnicas referidas em 2.2, de modo a reduzir a sua ocorrência. Nenhuma das técnica de controlo, seja ela pessimista ou optimista se apresenta como agradável para o utilizador. A sua utilização deve ser sempre evitada, mantendo sempre que possível o paralelismo.

5.2 Estruturação da interacção

Na secção 4.3.2 foram identificadas as operações suportadas pelo modelo de suporte. Interessa agora estudar a forma como estas podem introduzir situações de conflito que devem ser evitadas e resolvidas. Neste estudo classificam-se em três classes:

- **Operações atómicas**. Incluem-se nesta classe, as operações de alteração das propriedades recorrendo a um *click* sobre o monitor de propriedade;
- **Operações de estruturação**. Inclui as Conexões. São operações de associação de Contentores realizadas em duas fases, tipicamente num muito curto espaço de tempo;
- **Operações de edição**. Inclui as operações de edição de Conteúdos. São operações duradouras realizadas em duas fases.

Recorde-se que a distinção entre os dois últimos tipos de operação foi apresentada na matriz da Figura 16.

Interessa agora identificar técnicas de controlo que permitam resolver os problemas levantados por cada uma delas, tendo em conta a natureza de cada uma:

- **Operações atómicas**. São naturalmente serializadas e executadas, não necessitando de controlo;
- **Operações de estruturação**. A ocorrência de conflitos gerados por estas operações é pequena, excepto quando são feitas sobre objectos que mantêm Conteúdos simples em edição. Como tal, recorre-se a uma técnica semi-optimista de controlo da concorrência. O utilizador vai realizando a operação até ao momento em que é detectado um conflito. Se tal acontecer a execução é abortada ou é reposto o estado inicial, respectivamente se acção está em curso ou se já foi terminada;
- **Operações de edição**. Dá-se aos utilizadores a liberdade de editar os Conteúdos livremente. Quando são detectados conflitos e terminadas as operações de edição, promove-se a escolha de uma versão pelo grupo. Durante a fase de resolução proíbe-se a edição, torna-se o controlo momentaneamente pessimista.

5.3 Objectos

Nesta secção são apresentados os mecanismos que implementam as técnicas apresentadas, sobre o modelo de suporte. São isolados inicialmente os problemas de controlo na edição de Conteúdos simples. Finalmente, constroem-se sobre estes os objectos que permitem controlar as operações de estruturação.

Os mecanismos referidos são introduzidos como uma extensão ao modelo. Em cada objecto de partilha de dados (Contentores e Conteúdos) é introduzido um novo componente:

- **Controlador**. Controla o acesso aos objectos realizando a detecção e resolução de conflitos.

5.3.1 Controlo de operações de edição

Como foi dito, as operações de edição são controladas segundo uma técnica optimista na manipulação dos dados. Na realidade o controlo é misto de forma a utilizar a fase de controlo inicial para efectuar a monitorização.

A Figura 25 apresenta uma máquina de estados que representa o funcionamento do controlador de Conteúdos simples.

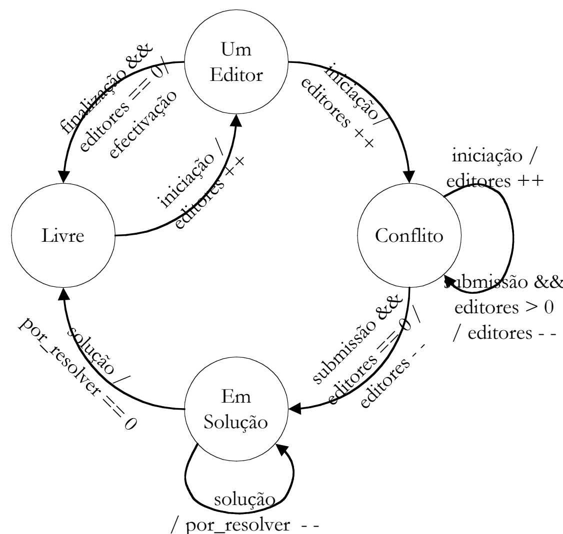


Figura 25 – Estados para edição concorrente de Conteúdos simples

Observam-se quatro estados:

- **Livre**. O objecto não está a ser acedido. A distinção entre esta fase e as restantes mostrar-se-á relevante no controlo das operações de estruturação.
- **Um Editor**. O objecto está a ser acedido sem conflitos. O acesso às réplicas do objecto é livre, podendo os utilizadores iniciar as suas edições normalmente.
- **Conflitos**. Foi detectado um conflito. O acesso às réplicas continua a ser livre (continuam a ser admitidas iniciações de operações de edição), sendo no entanto consciencializados os utilizadores do facto de estarem a entrar em conflito.
- **Em Solução**. Foram terminadas todas as operações em conflito. As réplicas devem apresentar uma solução para o conflito. A solução pode ser determinada pelo utilizador ou programada na réplica. No final, as soluções apresentadas conduzem a um estado coerente das réplicas do objecto. Durante esta fase o acesso do utilizador às operações de edição é negado.

Note-se que os dois primeiros estados estão sincronizadas entre os objectos, mas a transição do terceitro para o quarto é realizada para cada objecto em particular, voltando a sincronizar-se na transição do último para o primeiro.

5.3.2 Controlo de operações de estruturação

O controlo das operações de estruturação é feito em comunidades de objectos e não sobre um objecto apenas. De facto, uma operação deste tipo afecta não só os Contentores envolvidos como também as estruturas em que se inserem, pelo que é feita a detecção de conflitos sobre conjuntos de objectos.

Uma Conexão só é possível se:

- No momento da iniciação, todos os objectos que são directamente referenciáveis para cima e para baixo na hierarquia em que se encontra a Origem estiverem livres;
- No momento da finalização, todos os objectos que são directamente referenciáveis para cima e para baixo na hierarquia em que se encontra o Destino estiverem livres.

Considera-se que um objecto está livre se não for a Origem de alguma Conexão em curso ou se for um Conteúdo simples que não esteja a ser editado (estado Livre).

A Figura 26 representa a visibilidade para detecção de conflitos na manipulação de um objecto.

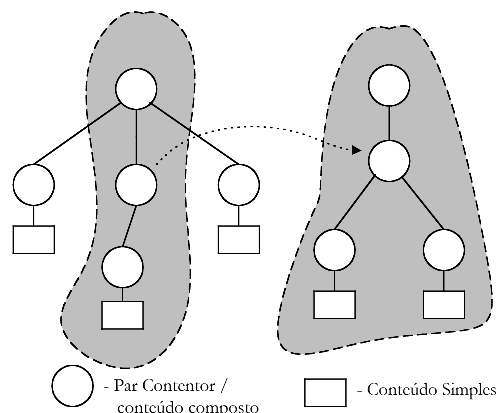


Figura 26 – Detecção de conflitos em operações de estruturação de dados

Ao nível da interface utilizador, não são reflectidos os tempos de determinação de possibilidade ou impossibilidade de estabelecimento de uma Conexão. A manipulação é livre até que um conflito seja detectado. Nesse caso a Conexão é abortada. Considera-se pois que esta técnica é semi-optimista.

Como vimos, a detecção de um conflito pode ocorrer em dois momentos: na Iniciação ou na Finalização. Caso o conflito seja detectado logo na Iniciação, não deve ser abortada a operação de imediato. O utilizador deve monitorizar o facto, mas não deve ser impedido de terminar a interacção. No final, repõe-se a situação inicial sem retirar o controlo do utilizador.

5.4 Interface Utilizador

Tal como foi referido para a componente de suporte à interacção, o modelo apresentado não restringe a componente de interface utilizador, sendo esta desenhada para cada aplicação. Importa mesmo assim, tecer algumas considerações sobre esta componente e introduzir um conjunto de propostas coerente com as apresentadas no Capítulo 4.

Na apresentação das técnicas de controlo da concorrência foram identificadas várias fases, que podem ser directamente monitorizadas através de objectos Monitores. As secções seguintes abordam esta facilidade.

5.4.1 Monitorização da concorrência na edição de conteúdos simples

Na descrição da técnica de controlo deste tipo de operação identificaram-se quatro estados que podem ser directamente monitorizadas:

- **Livre e Um Editor**. Não existem conflitos pelo que não deve ser feita qualquer monitorização de modo a não introduzir elementos distractivos na interface;
- **Conflitos**. Detectaram-se conflitos. Apresenta-se um monitor junto dos Contentores sobre os quais estejam a ser feitas edições de Conteúdos simples. Sempre que for feita uma iniciação de edição de um Conteúdo que conduza à passagem a esta fase também se apresenta o Monitor. Na aplicação apresentada no Capítulo 7, recorreu-se à metáfora do semáforo para fazer a monitorização. Esta fase é monitorizada pelo semáforo amarelo (ver Figura 27).
- **Em Solução**. Os conflitos devem ser resolvidos e não são permitidas edições ao Conteúdo. Proíbe-se a edição e promove-se a apresentação de uma solução. A monitorização pode ser feita por um semáforo vermelho, tal como apresentado na Figura 27. A transição da fase 4, novamente para a fase 1 pode ser feita mantendo o monitor, fazendo transitar o estado do semáforo para verde durante alguns segundos, apagando-o logo de seguida.

Figura 27 – Estados do monitor de concorrência

5.4.2 Monitorização da concorrência na estruturação de Contentores

O controlo deste tipo de interacção também é faseado. Identificam-se duas fases:

- **Iniciação**. Se durante esta fase for detectado algum conflito deve colocar-se junto da Origem o monitor de concorrência no estado vermelho. O utilizador mantém o controlo mas é desmotivado de continuar o processo de Conexão, sendo este abortado logo que libertado o botão do rato;
- **Finalização**. Se na execução desta fase for detectado algum conflito deve também colocar-se o monitor de concorrência no estado vermelho, junto do Destino. A Conexão não é concluída. O monitor deve manter-se durante alguns segundos desaparecendo depois.

Parte III

REALIZAÇÕES

Capítulo 6

REALIZAÇÃO DO MODELO DE INTERACÇÃO

Neste Capítulo descreve-se uma implementação do modelo descrito nos Capítulos anteriores. Apresenta-se o modelo de objectos, acompanhado da descrição das classes e métodos relevantes, assim como um diagrama de sequência que descreve as invocações realizadas entre os objectos do modelo.

6.1 Modelo de objectos de suporte

Na Figura 28 pode-se observar o diagrama UML de classes que representa a implementação do modelo de suporte à interacção.

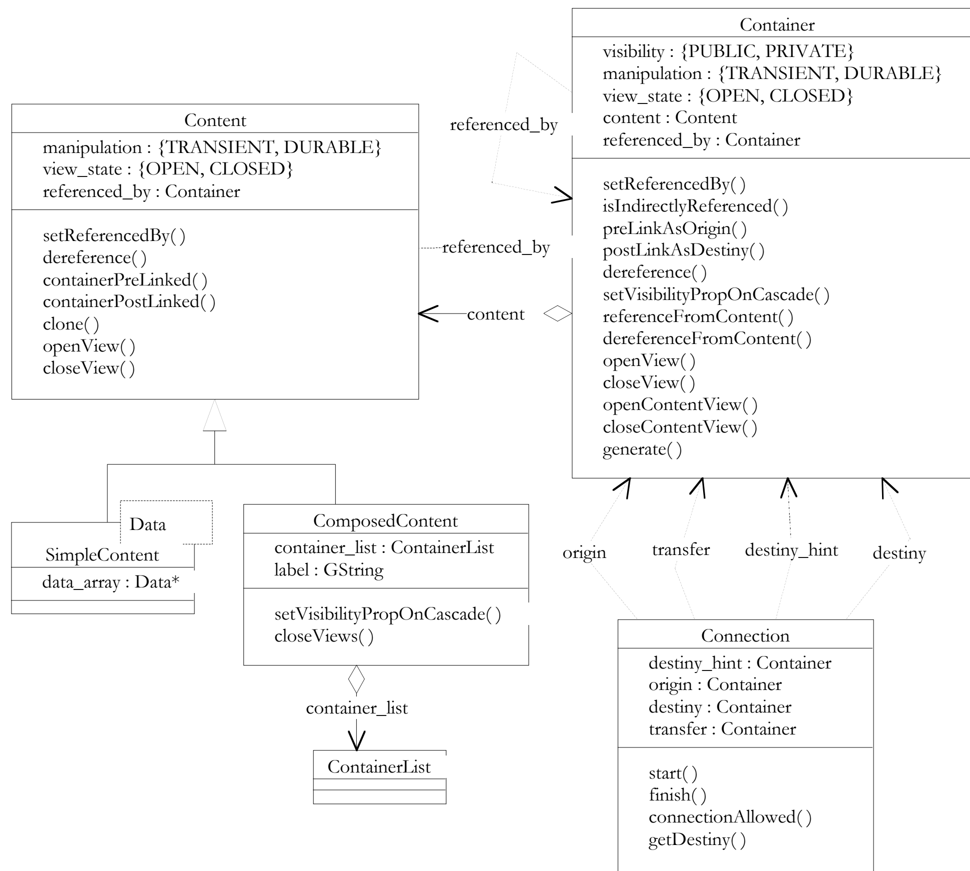


Figura 28 - Modelo de objectos de suporte à interacção

Os componentes do diagrama de classes são:

- **Container**. Classe correspondente aos objectos de interacção do género Contentor. As propriedades dinâmicas de visibilidade e manipulação enunciadas na descrição do modelo permitem definir as operações de estruturação da interacção a executar. Os métodos `preLinkAsOrigin` e `postLinkAsDestiny` indicam que o contentor participa, respectivamente, como origem ou destino de uma conexão. O método `generate` implementa a funcionalidade do objecto contentor criar ou não outros contentores (excertos relevantes do código podem ser consultados no Apêndice C – Figura 67 e 69);
- **Content**. Implementa os objectos de interacção do modelo designados por Conteúdos. Os objectos desta classe armazenam a informação relativa aos Contentores a que pertencem (ver Figura 69). Tem duas classes derivadas:
  - **SimpleContent**. Classe representativa de um Conteúdo Simples. Suporta a informação definida ao nível da aplicação;
  - **ComposedContent**. Representa os objectos Conteúdo formados por um conjunto de Contentores. Implementa a estrutura hierárquica definida pelo modelo de interacção;
- **Connection**. Classe de objectos de interacção do tipo Conexão. Recebem por estímulo da aplicação os eventos de início e fim de interacção e a identificação dos Contentores envolvidos. Avaliam, através do método `connectionAllowed`, a validade da conexão e despoletam as operações de interacção a realizar (ver Figura 70).

Segue-se o diagrama de sequência de invocações realizadas pelos objectos pertencentes ao modelo de interacção. Este diagrama não é exaustivo, interessa apenas mostrar as principais invocações que decorrem a partir de uma interacção:

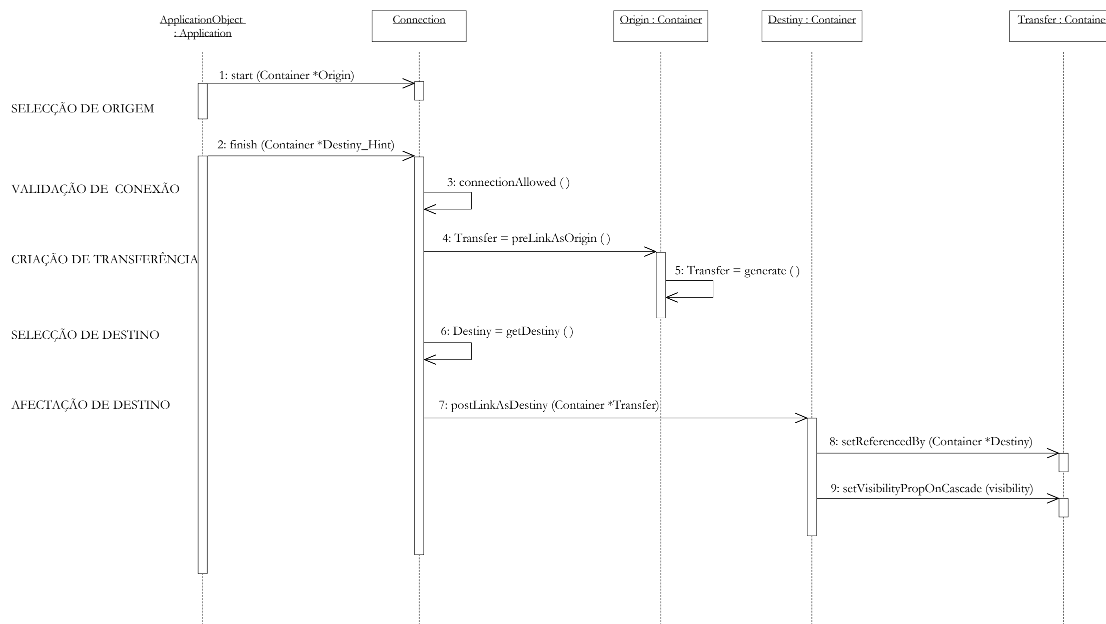


Figura 29 - Diagrama de Sequência

Repare-se que na execução de uma interacção participam três objectos distintos da classe Container: contentores *Origin*, *Transfer* e *Destiny*. Assim, uma conexão envolve a colaboração com os três Contentores referidos, denominados a partir deste ponto, Origem, Transferência e Destino.

As fases de iniciação e finalização de uma Conexão não são directamente representadas pelos métodos `start` e `finish`. A validação de uma Conexão é realizada após o conhecimento da Origem e do Destino, e

a criação da Transferência é feita com a garantia de validade da interacção e não de forma provisória, isto é, uma Transferência só é criada, se necessário, após ser conhecido o destino (de outro modo implicava operações de *undo* desnecessárias).

Seguindo o diagrama, descrevem-se as etapas que compõem a sequência de invocações:

- **Selecção de Origem**. Nesta fase a aplicação ou objecto da aplicação detecta qual o Contentor onde se iniciou a interacção. A conexão é informada do contentor origem da interacção por invocação do método `start`;
- **Validação da Conexão**. A interacção está concluída tendo a aplicação conhecimento do contentor que participou no final da interacção. A conexão tem, através da invocação do método `finish`, acesso a esse Contentor e pode certificar-se da validade da interacção. O método `connectionAllowed` cumpre essa validação e, normalmente será redefinido ao nível da aplicação para englobar as restrições específicas;
- **Criação da Transferência**. O contentor origem é responsável pela criação do contentor transferência, ou seja, comporta-se como um objecto que desempenha uma determinada funcionalidade atendendo às propriedades que apresenta no início da interacção. Pode, como exemplos, gerar uma cópia sua mantendo ou não o seu Conteúdo, gerar outro tipo de Contentor ou retornar-se a si próprio recorrendo à especialização dos Contentores e redefinição dos métodos `preLinkAsOrigin` e `generate`;
- **Selecção de Destino**. Habitualmente, numa interacção, o contentor final identifica o contentor destino da Conexão, em algumas interacções a aplicação pode entender que o contentor final representa apenas uma "pista" para o destino. O método `getDestiny`, quando redefinido, pode implementar a selecção de Contentores Destino por via indirecta;
- **Afectação de Destino**. O Contentor Destino está determinado e, por invocação do método `postLinkAsDestiny`, recebe a Transferência. De acordo com as propriedades dinâmicas que apresente o contentor Destino (e/ou o seu Conteúdo) pode, por exemplo, passar a conter o Contentor Transferência, receber os dados presentes no Conteúdo da Transferência ou executar funcionalidades adicionais por enriquecimento do método em classes *Container* especializadas.

Desta descrição conclui-se que a fase de Iniciação de uma Conexão engloba as etapas de **Selecção de Origem**, **Validação de Conexão** e **Criação de Transferência**.. As restantes etapas respondem pela fase de Finalização.

6.2 Modelo de objectos de controlo

No Capítulo 5 foi introduzida a componente denominada Controlador sobre os objectos de partilha de dados: os Contentores e os Conteúdos simples. A sua implementação abstracta recorre à utilização da *framework* de concorrência. Para tal foram criada uma nova composição que respeita os requisitos de não transparência dos objectos. Como vimos, as fases dos processos de controlo da concorrência são monitorizadas pelo que a interface das composições deve apresentar os métodos de Iniciação e Finalização das invocações (*preControl e postControl*).

A nova composição é:

- ***Opaque_Passive_Sync_Rec_Inter***. Composição não transparente de sincronização e recuperação de objecto. Os seus métodos e dados são públicos de modo a poderem ser directamente invocados através das operações de estruturação da interacção e poderem ser monitorizados.

A partir destas composições, a componente Controlador é implementada pelo padrão *proxy* tal como foi apresentado em 3.1.1.1.4.

A Figura 30 apresenta a comunidade de classes que implementa o controlo da concorrência:

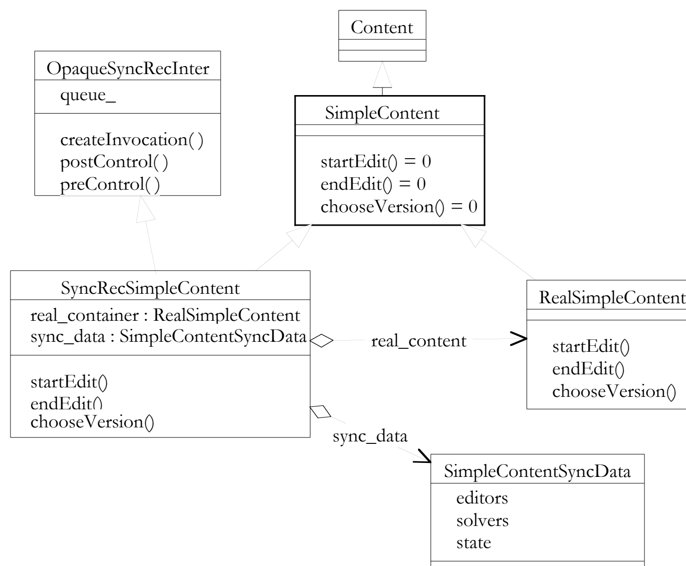


Figura 30 - Controlo de concorrência na edição de Conteúdos

Classes envolvidas:

- ***SimpleContent***. Representa o Conteúdo Simples de forma abstracta.
- ***RealSimpleContent***. Implementa o Conteúdo Simples sem controlo de concorrência. O método *startEdit()* não altera os dados, o método *endEdit()* actualiza os dados e o método *chooseVersion()* permite escolher uma versões submetidas.
- ***SyncRecSimpleContent***. Implementa o controlo da concorrência, servindo-se dos dados mantidos pelo ***SimpleContentSyncData***, mapeando os métodos de *startEdit()* e *endEdit()* directamente em *preControl()* e *postControl()*. Depois de efectuar o controlo de acesso, invoca os métodos correspondentes do *real_container*. Obviamente as *invocations* são guardadas entre o *startEdit()* e o *endEdit()*. O controlo é misto por ser feito antes e depois da manipulação. O controlo inicial serve apenas para monitorização, não sendo pois considerado ao nível da partilha de dados, pelo que se considera a política como optimista.

Excertos do código destes métodos é apresentado na do Apêndice C (Figura 71).

Capítulo 7

EXEMPLO DE APLICAÇÃO *IDEAGEN*

Descreve-se agora uma aplicação construída sobre a plataforma descrita no Capítulo anterior. Implementa um subconjunto do processo de geração de ideias Brainstorming e um conjunto de mecanismos de comunicação em grupo.

7.1 Enquadramento

Com o objectivo de instanciar e experimentar as soluções introduzidas na Parte II, construiu-se uma aplicação que implementa a fase de geração de ideias do processo de facilitação de reuniões *Brainstorming*.

A aplicação *IDEAGEN* foi construída por reengenharia da aplicação *NgMeeting* (ver 3.2.1) e suportada pela biblioteca apresentada no Capítulo anterior.

A construção da aplicação implicou a realização das seguintes tarefas:

- Conversão do servidor de *MBus* para plataforma *Unix/Solaris* em C++;
- Redesenho da aplicação *NgMeeting*, integrando-lhe novas funcionalidades e retirando-lhe outras;
- Integração do modelo na nova aplicação, reconvertendo o modelo de objectos da aplicação;
- Integração do controlo de concorrência num nó central.

Os passos aqui apresentados serão detalhados ao longo das duas secções seguintes.

Exemplos de utilização da aplicação são apresentados no Apêndice B. Os exemplos ilustram o modelo até aqui descrito e apresentam a aplicação.

7.2 Requisitos

Pretendeu-se com esta aplicação apresentar um conjunto muito simples de funcionalidades com requisitos baseados nas preocupações deste trabalho.

Os requisitos apresentados não são pois ao nível da aplicação concreta, mas sim ao nível das experiências e do estudo que se pretendeu realizar:

- Integrar modelos optimistas e semi-optimistas de controlo da concorrência, com o respectivo suporte de recuperação;
- Implementar diversos mecanismos de monitorização das actividades de grupo, nomeadamente os respeitantes ao modelo de interacção;
- Relaxar a coerência de dados e visual, segundo os modelos *WYGIWIG* e *WYSIWIMS*.

- Exercitar as características relevantes do modelo:
    - Estruturação hierárquica da informação;
    - Manipulação dinâmica de propriedades;
    - Edição de Conteúdos;
    - Modelo de Conexões.

Para tal, enriqueceu-se a aplicação com as seguintes funcionalidades;

- Estruturação hierárquica de ideias em "pastas". Cada "pasta" mantém um conjunto de ideias e "pastas";
- Introdução de um mecanismo de comunicação do tipo *Talker* para troca de mensagens dentro do grupo;
- Introdução de um mecanismo de comunicação para troca de mensagens com monitorização da sua leitura (*Confirm*).

Para simplificar foram retiradas do processo de facilitação as fases de geração de comentários e de votação das ideias.

A plataforma de desenvolvimento manteve-se, sendo utilizadas as *Microsoft Foundation Classes*, o EdGar (ver 3.1.3.1), o *MBus* (ver 3.2.1.2.1). O controlo da concorrência foi suportado pela *framework* de concorrência (ver 3.1.1.1)

7.3 Desenho

Como foi dito o *IDEAGEN* foi construído com base no *NgMeeting*, pelo que foi mantida a sua estrutura na generalidade, sendo substituída a componente de suporte à interacção pela apresentada no modelo.

A descrição da aplicação vai focar-se essencialmente nos aspectos de desenho dessa componente, sendo abordada superficialmente a arquitectura de objectos herdada do *NgMeeting*. Uma descrição mais pormenorizada desta pode ser encontrada em [Ho 96].

7.3.1 Funcionalidades do IDEAGEN

Nesta secção são apresentados alguns aspectos funcionais do IDEAGEN, nomeadamente os três mecanismos fundamentais identificados: geração de ideias, comunicação via *Talker* e comunicação via *Confirm*.

Para enquadrar o texto apresentado neste capítulo apresenta-se (na Figura 31) uma imagem do funcionamento da aplicação.

7.3.1.1 Geração e manipulação de ideias

A geração de ideias é realizada no espaço privado de cada participante da reunião. O participante, no seu espaço privado, é livre de estruturar as ideias e preencher o seu conteúdo do modo que entender.

A estruturação das ideias obedece a uma forma hierárquica, no topo da hierarquia terá que existir obrigatoriamente uma "pasta" de ideias e em níveis inferiores poderão existir não só ideias, mas também, outras "pastas" com ideias. Tal assemelha-se à estrutura hierárquica de um editor de texto com diversas secções, subsecções e conteúdos textuais ou à estrutura de um sistema de ficheiros com os seus directórios e sub directórios.

Como foi referido, os contentores apresentam uma propriedade dinâmica designada por manipulação. No caso de uma ideia (subentende-se Contentor ideia) apresentar-se com manipulação durável, uma interacção de *drag'n'drop* levará à criação de uma cópia. Se a propriedade de manipulação for transitória

ocorre uma movimentação, apenas visual se se mantiver na mesma pasta ou estrutural se o movimento a colocar noutra pasta. Estas operações serão análogas às operações de *copy* e *move* realizadas em *browsers* de sistemas de ficheiros (no entanto, sem ser necessário recorrer a teclas para desambiguar algumas interacções).

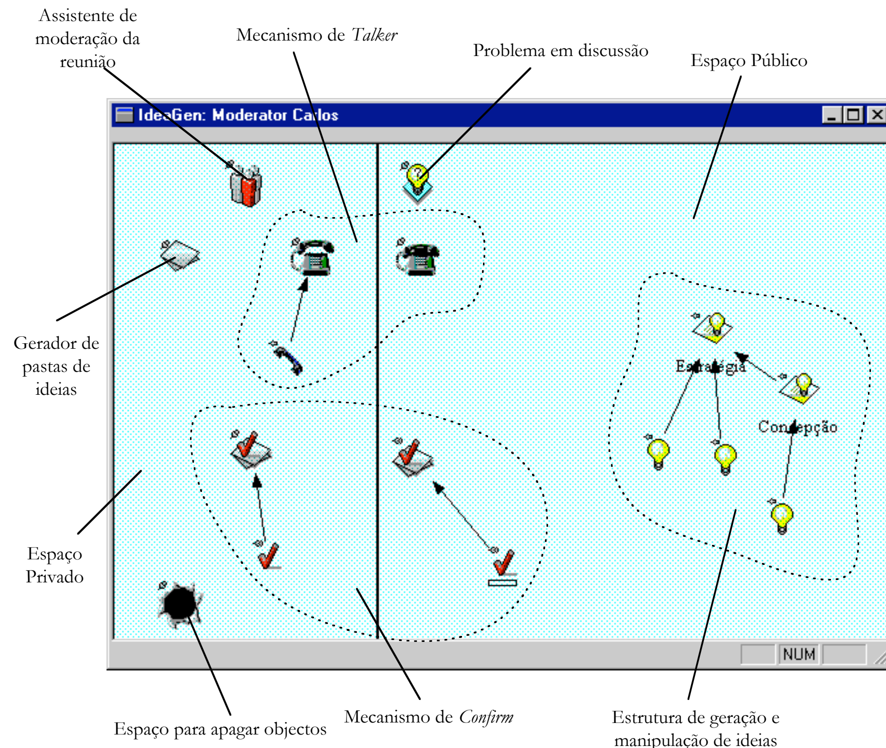


Figura 31 – Interface gráfica do IDEAGEN

A riqueza da estruturação é estendida com a possibilidade de realizar associações hierárquicas entre "pastas" de ideias, o equivalente num editor de texto, a escrever uma secção principal e uma subsecção em paralelo para no fim as integrar. Atente-se a esta última afirmação: o trabalho em paralelo ou a cooperação são favorecidos por esta forma de estruturação (divisão de tarefas distintas e integração natural da informação produzida).

Após a geração privada das ideias o participante tem a possibilidade de tornar a estrutura de ideias visível aos restantes elementos do grupo, bastando para tal colocá-la no espaço público. Neste espaço todos os participantes podem editar os conteúdos, alterar a estruturação das ideias e as associações entre "pastas" de ideias.

7.3.1.2 *Talker*

O mecanismo de *Talker* tem como funcionalidade permitir a troca de mensagens entre os participantes na reunião. As mensagens são produzidas no espaço privado de cada participante a partir do contentor *Talker* ali presente. São difundidas aos restantes membros do grupo quando os contentores correspondentes a cada mensagem são colocados sobre o contentor *Talker* existente no espaço público.

As mensagens são visualizadas numa janela de diálogo que surge sobre o contentor *Talker* público e a ordem de mostragem das mensagens pode seguir duas políticas: uma política optimista ou uma política pessimista.

Na política pessimista todos os participantes observam exactamente a mesma ordem de mensagens, ou seja, ocorre uma serialização das mensagens difundidas. A política optimista parte do princípio que o utilizador utilizou como contexto as mensagens recebidas até ao momento em que difunde a sua própria mensagem e, como tal, pode visualizar imediatamente a sua mensagem na janela de diálogo pública. Está-se perante um relaxamento da coerência visual, uma vez que, cada participante pode observar ordens particulares de mensagens. A política de mostragem de mensagens pode ser alterada dinamicamente.

7.3.1.3 Confirm

O mecanismo de *Confirm* apresenta uma funcionalidade semelhante ao mecanismo anterior, permite também a troca de mensagens, com a criação e difusão de mensagens a seguirem procedimentos semelhantes. As mensagens geradas por este mecanismo destinam-se a garantir o aviso de recepção, isto é, o participante que enviou esta mensagem pode observar por intermédio de um monitor o número de recepções confirmadas.

Assim um participante que receba uma mensagem através deste mecanismo deve abrir o seu conteúdo de forma a confirmar a sua recepção. Deste modo, o participante emissor vai observar no monitor correspondente a confirmação dessa recepção. Caso contrário o emissor apercebe-se que algum ou alguns dos participantes na reunião não confirmaram a recepção da sua mensagem.

7.3.2 Arquitectura

A arquitectura da aplicação segue no geral o que foi descrito para o *NgMeeting* em 3.2.1.1 e 3.2.1.2. A introdução do controlo da concorrência introduziu um novo elemento: um nó central, onde são analisados os conflitos e geridos os mecanismos de recuperação.

Este controlo poderia ser feito directamente sobre as réplicas dos objectos partilhados, no entanto, essa solução implicaria a sincronização das réplicas dos objectos que implementam o controlo, tal como descrito na apresentação da *framework* de concorrência. A resolução deste problema é resolvida por outros aspectos estudados no âmbito do projecto DASCo e não se considerou um problema a resolver no contexto deste trabalho.

Considera-se pois, uma arquitectura distribuída replicada, onde cada objecto se encontra replicado em cada nó da aplicação e ainda num Nó de Controlo (ver Figura 32).

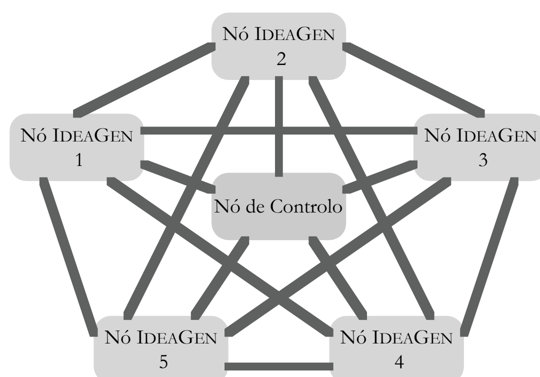


Figura 32 - Arquitectura conceptual

A utilização de políticas pessimistas de controlo da concorrência permitiria fazer toda a comunicação através do Nó de Controlo, no entanto continua-se a manter a comunicação distribuída de modo a gerar o paralelismo. As operações efectuadas sobre os dados partilhados são comunicadas a todos os restantes Nós

(incluindo o de controlo). Na presença de conflitos o Nó de Controlo encarrega-se de comunicá-los a todos os Nós que os deverão resolver.

O *MBus* implementa a comunicação distribuída recorrendo a um servidor central tal como descrito em 3.2.1.2.1. Por esta razão decidiu-se, para efectivar a comunicação integrar o Nó de Controlo com o servidor de *MBus*. O servidor denomina-se *IDEASERVER*.

A Figura 33 apresenta a arquitectura descrita, tal como foi implementada, descrevendo-se ainda um processo comum de distribuição.

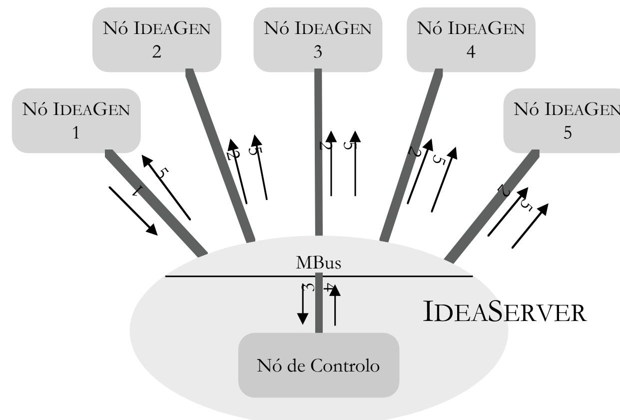


Figura 33 - Arquitectura real

Identificam-se cinco fases:

1. Um Nó (na figura o Nó 1) envia um evento para distribuir pelas réplicas;
2. O evento é reflectido para os Nós seleccionados na mensagem (na figura, todos os restantes);
3. O evento é comunicado internamente ao Nó de Controlo;
4. Caso seja detectado um conflito, ele é enviado ao serviço de distribuição;
5. O serviço distribui o evento por todos os Nós.

Se fosse necessário distribuir o controlo da concorrência, sincronizando o acesso às réplicas o sistema funcionaria de forma idêntica, abolindo o Nó de Controlo e a comunicação por ele gerada. Todas as réplicas fariam por si a detecção e comunicação de conflitos internamente.

7.3.3 Modelos de objectos

Nesta secção descrevem-se as classes, métodos e suas relações que se consideram mais importantes na realização do *IDEAGEN*.

7.3.3.1 IDEAGEN

A construção da aplicação IDEAGEN foi feita sobre um núcleo de objectos já utilizados na construção das aplicações *NgTool* e *NgMeeting*. Mantém-se um conjunto pequeno de objectos que realizam a gestão das operações realizadas pelo utilizador do Nó e recebidas pelo sistema de distribuição de mensagens. A Figura 34 apresenta o esse conjunto de classes e as suas relações.

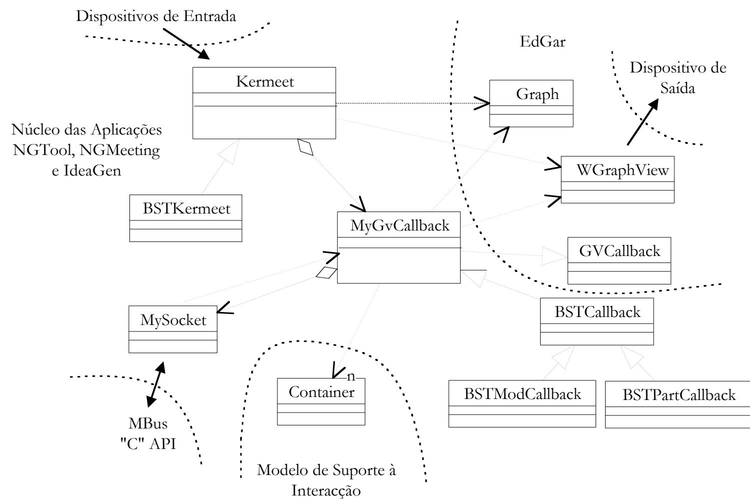


Figura 34 - Núcleo de objectos do IdeaGen

Observa-se a ligação do núcleo de objectos apresentado com as plataformas utilizadas, nomeadamente o EdGar, o *MBus* e o modelo de suporte à interacção.

A funcionalidade das classes apresentadas é a seguinte:

- ***Kermeet*** e ***BSTKermeet***. Recebem os eventos provenientes dos dispositivos de entrada que filtram e delegam em *MyGVCallback*. Responsabilizam-se por fasear as reuniões. *BSTKermeet* implementa o comportamento das reuniões de geração de ideias (único tipo de facilitação permitido pelo IDEAGEN);
- **MyGVCallback**, **BSTCallback**, **BSTModCallback** e **BSTPartCallback**. Processam todos os eventos provenientes dispositivos de entrada. Recebem e geram os eventos distribuídos através do serviço de comunicação do *MBus* através da classe **MySocket**. Para tal baseieam-se na interface, disponibilizada pelo EdGar na classe **GVCallback**. As classes que derivam de *MyGVCallback* implementam os comportamentos de resposta a eventos nos tipos de reuniões definidas nas três aplicações, e mais especificamente, o comportamento do moderador e dos participantes ordinários. Mantêm a estrutura de suporte da interacção numa tabela de Contentores. São ainda responsáveis por utilizar a classe *WgraphView* para tratar a interface utilizador.
- **Graph** e **WGraphView**. São as classes do EdGar responsáveis por manter respectivamente: as estruturas de grafos e o seu contexto de visualização. Sobre elas são construídas as representações dos Contentores, como veremos de seguida;

7.3.3.1.1 Instanciação do modelo de suporte à interacção

Para instanciar e adaptar o modelo na aplicação IDEAGEN recorreu-se essencialmente às duas classes representadas ao centro na Figura 35.

A classe **EdgarNodeContainer** (Apêndice C - Figura 72 e 74) apresenta a adaptação da componente de apresentação recorrendo ao EdGar. Os Contentores apresentam uma interface icónica suportada pela classe **NodeViewPtr**. Na classe **BSTComposedContent** é definida a apresentação dos conjuntos de contentores recorrendo a objectos do tipo **LinkViewPtr**. A manutenção destes objectos é feita nos próprios contentores de forma a distribuir o seu controlo. Cada Contentor mantém a sua apresentação no contexto do Conteúdo composto onde se encontra.

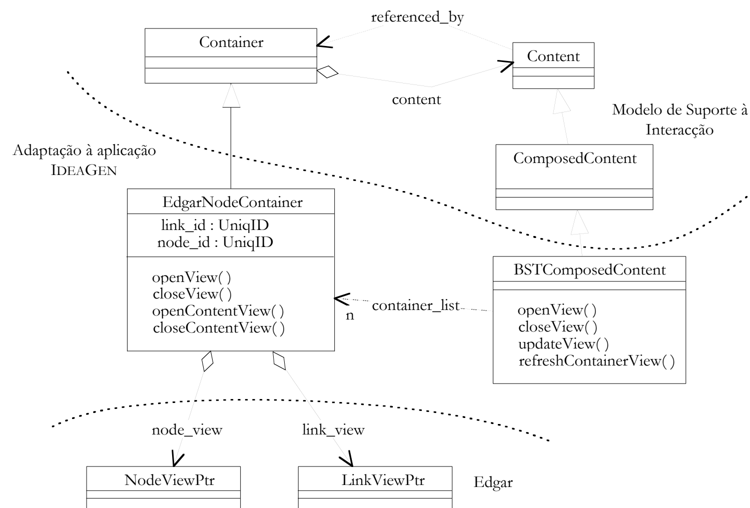


Figura 35 - Instanciação do modelo de suporte à interacção no IdeaGen

A partir da adaptação apresentada para suporte da interacção definiram-se classes próprias para cada tipo de Contentor e Conteúdo simples onde são suportadas as funcionalidades da aplicação. As secções seguintes apresentam esses tipos.

Foi ainda derivada uma classe de *Connection* : a classe **BSTConnection** (Figura 74) que redefine os métodos *start* e *finish* de modo a induzir nos restantes objectos o seu comportamento gráfico. É ainda redefinido o método *connectionAlowed* de modo a implementar as restrições de nível aplicação no que diz respeito às Conexões permitidas.

Foram também utilizados todos os tipos de monitores apresentados nos Capítulos 4 e 5.

7.3.3.1.2 Contentores

Existem duas classes de Contentores que não são derivadas de *EdgarNodeContainer* porque o seu Conteúdo composto não é representado da mesma forma: **PrivateSpaceContainer** e **PublicSpaceContainer**. Estas mantêm os Contentores que visualmente não mantêm nenhuma ligação com outro que seja hierarquicamente superior. Desta forma os Contentores públicos e privados são mantidos em conjuntos separados, sendo estes espaços encarados eles próprios como contentores.

Os restantes Contentores são derivados de *EdgarNodeContainer* e são agrupados em três grupos de funcionalidades: geração e manipulação de ideias, geração e manipulação de mensagens de *talker* e geração e manipulação de mensagens de *Confirm*.

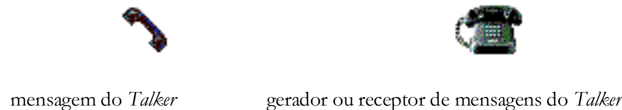
Relativos às ideias (ver Figura 36):

- **SheetGenContainer**. Gerador privado de “pastas” de ideias;
- **SheetContainer**. “Pasta” de ideias. Gere e estrutura ideias e outras “pastas”;
- **IdeaContainer**. Uma ideia;

Figura 36 - Contentores para geração e manipulação de ideias

Relativos ao mecanismo de comunicação *Talker* (ver Figura 37):

- **TalkGenContainer**. Gerador privado de mensagens para o *Talker*;
- **TalkMsgContainer**. Mensagem do *Talker*;
- **TalkRecipientContainer**. Receptor público para recepção de mensagens do *Talker*.

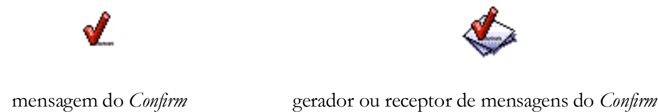


mensagem do *Talker* gerador ou receptor de mensagens do *Talker*

Figura 37 - Contentores para geração e manipulação de mensagens do *Talker*

Relativos ao mecanismo de comunicação *Confirm* (ver Figura 38):

- **ConfirmGenContainer**. Gerador privado de mensagens para o *Confirm*;
- **ConfirmMsgContainer**. Mensagem do *Confirm*;
- **ConfirmRecipentContainer**. Receptor público para recepção de mensagens do *Confirm*.



mensagem do *Confirm* gerador ou receptor de mensagens do *Confirm*

Figura 38 - Contentores para geração e manipulação de mensagens de *Confirm*

Relativos a outros Contentores (ver Figura 39):

- **HoleContainer**. Contentor com Conteúdo nulo que permite apagar outros Contentores recorrendo a Conexões.
- **ProblemContainer**. Contém o problema em discussão.
- **ModeratorContainer**. Implementa o faseador da reunião no espaço privado do seu moderador da reunião. Permite gerir as diversas fases das reuniões.

buraco para remoção de Contentores problema em discussão faseador do moderador

Figura 39 - Outros Contentores com funcionalidades diversas

7.3.3.1.3 Conteúdos simples

No IDEAGEN foram utilizadas duas classes derivadas de *SimpleContent*:

- **TextSimpleContent**. É um Conteúdo simples de texto que pode ser editado. Utilizado para manter os Conteúdos dos diversos tipos de mensagens e das ideias.
- **LinearStateMachineContent**. Implementa um Conteúdo que é uma máquina de estados linear. Utilizado no faseador da reunião.

7.3.3.2 IDEASERVER

A arquitectura de dados do IDEASERVER baseia-se em três módulos principais:

- **Comunicação**. Gera a interface com *MBus* reflectindo as mensagens recebidas e delegando-as ao Nó de Controlo. Gere os utilizadores e os canais de comunicação com eles.

Os dois módulos restantes implementam o Nó de Controlo.

- **Gestão de Dados**. Mantém a referência sobre os dados que implementam o controlo da concorrência.
- **Dados Partilhados**. Mantém os dados partilhados, implementando controlo de concorrência sobre eles.

As classes de objectos relevantes a estes módulos são apresentadas na Figura 40.

Aspectos relevantes das classes representadas:

- **LocalServerProxy**. Serve de interface para as mensagens recebidas pela rede. Faz o seu despacho para os utilizadores e delega o seu tratamento às componentes de controlo;
- **RemoteUserProxy**. Representa os objectos dos Nós da aplicação, tratando da comunicação de forma transparente. Existe um por cada Nó;
- **User** e **Users**. Gerem a informação sobre os utilizadores ligados em cada sessão, utilizando os *RemoteUserProxy*'s para lhes enviar mensagens.
- **MeetingData**. Serve apenas para manter referências para os diversos subsistemas existentes na aplicação;
- **TalkData**, **ConfirmData** e **IdeaData**. Mantêm os dados partilhados de cada subsistema;
- **Serializer**. Objecto que atribui identificadores de ordem aos pedidos efectuados no mecanismo de *Talker* serializando-os;
- **ConfirmController**. Mantém uma lista de mensagens em trânsito, gerindo as suas confirmações.
- **IdeaController**. Gera uma estrutura que representa a hierarquia de objectos presentes no espaço público, controlando a sua manipulação segundo o modelo apresentado no Capítulo 5.

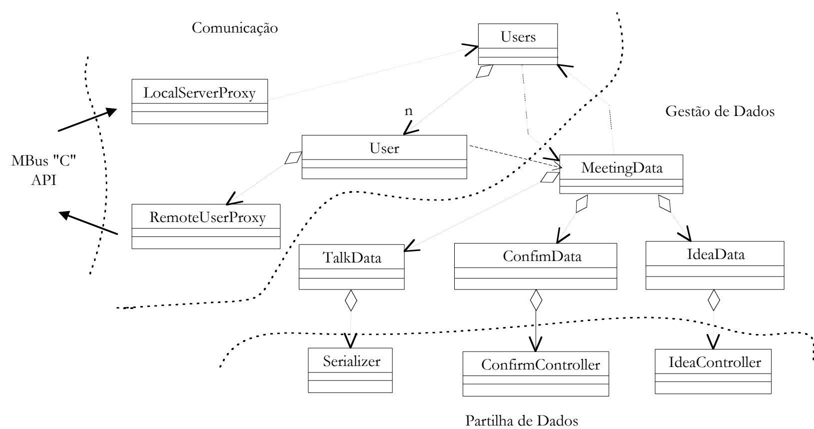


Figura 40 – Diagrama de classes do IDEASERVER

Os objectos de dados partilhados são controlados recorrendo ao padrão *proxy* através de objectos adaptados da *framework* de concorrência tal como foi apresentado em 6.2.

O servidor é multi-tarefa, separando as funcionalidades de distribuição das de controlo. A geração da concorrência apoiou-se nos mecanismos disponibilizados pelo ACE [Schmidt 95] e na implementação do padrão de geração da concorrência da *framework*.

7.3.4 Captura de acções de interacção

As acções de interacção identificadas em 4.3.2 foram utilizadas para estruturar a manipulação dos objectos no IDEAGEN. Tal como foi dito, é utilizado apenas o botão esquerdo do rato para realizar todas as acções identificadas.

Desta forma tornou-se necessário implementar sobre o sistema operativo um mecanismo de mais alto nível que permita distinguir as diversas entradas possíveis, nomeadamente: *click*, *double-click* e *drag'n'drop*. Para tal implementou-se a máquina de estados da Figura 41.

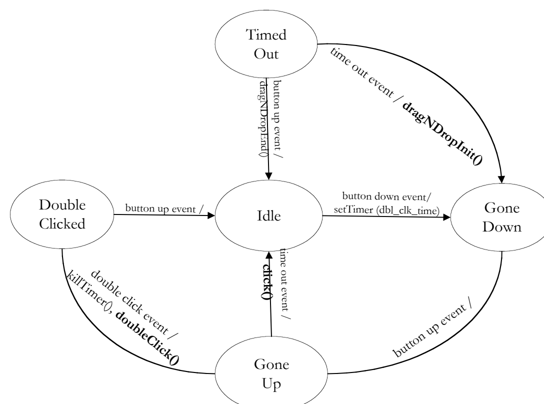


Figura 41 - Modelação da interacção a partir do rato

São reconhecidos os eventos de sistema: *button up*, *button down* e *double-click*. A partir destes e do tempo definido no sistema para um intervalo máximo permitido para consideração de um *double-click*, são identificadas as quatro primitivas de interacção:

- ***click***. Permite alterar as propriedades dinâmicas dos objectos quando realizada sobre um monitor de propriedade;
- ***doubleClick***. Permite abrir e fechar Conteúdos simples ou compostos;
- ***dragNDropInit***. Realiza a fase de iniciação de uma Conexão;
- ***dragNDropEnd***. Realiza a fase de finalização de uma Conexão.

Como se pode observar na máquina de estados a distinção entre as primitivas de interacção é feita recorrendo a um temporizador que uma vez expirado, permite identificar um *click*.

Capítulo 8

CONCLUSÕES E TRABALHO FUTURO

Apresenta-se finalmente, um conjunto de conclusões sobre o trabalho realizado e sobre soluções técnicas apresentadas. É ainda feita uma introdução de novos temas de estudo que podem ser objecto de trabalho futuro.

Considerou-se neste estudo a necessidade de identificar técnicas de suporte e controlo da interacção, adequadas à manipulação simultânea de um espaço de dados partilhado, como o problema genérico a resolver. Como pode ser depreendido pelo estudo até aqui apresentado, o problema é muito vasto. Como tal, importa agora tirar algumas ilações sobre as soluções apresentadas e apontar algumas direcções de trabalho futuro em tópicos que não foram abordados e que importa abordar em soluções para problemas mais abrangentes.

Em primeiro lugar propõem-se neste trabalho um conjunto de técnicas de análise que em conjunto, conduzem a especificações de desenho comuns a conjuntos de aplicações. Considera-se que as interacções podem ser duradouras e devem ser tratadas de forma não transparente pelos sistemas. Os requisitos de coerência visual e de dados devem ser relaxados sempre que possível. As incoerências devem ser detectadas e a probabilidade da sua ocorrência deve ser reduzida sempre que possível.

A partir das técnicas de análise indicadas, apresenta-se um conjunto adequado de soluções de desenho. **Advoga-se um arquitectura distribuída replicada que integre controlo optimista da concorrência. Este controlo deverá fornecer mecanismos de recuperação e ser monitorizado segundo os modos *WYGIWIG* e *WYSIWIMS*. Os dados deverão ser estruturados segundo uma organização hipertexto**.

Como vimos anteriormente, a utilização de técnicas de controlo optimista da concorrência permite resolver alguns aspectos negativos introduzidos sobre a interface utilizador pelos métodos pessimistas tradicionais. No entanto, outros, que se apresentam como menos graves, são levantados pelas técnicas aqui utilizadas, pelo que se conclui que **todas as técnicas de controlo da concorrência e resolução de conflitos, introduzem elementos desagradáveis na interacção com os utilizadores, pelo que deve ser evitada a sua utilização, mantendo e gerando o máximo de paralelismo possível. As soluções para este problema assentam na estruturação e separação da informação, relaxando os requisitos de visualização**.

A construção de um modelo de suporte e controlo da interacção, bem como a sua integração numa aplicação, permitiram testar e comprovar as soluções genéricas apontadas. No entanto, e principalmente no que diz respeito à componente de suporte existem algumas conclusões a tirar que resultam da utilização do sistema realizado. **O modelo introduz um conjunto de primitivas de utilização muito simples, com um conjunto de características comuns, aliado a uma semântica muito rica, induzida pelas propriedades definidas sobre os objectos e pela disponibilização dos mecanismos de monitorização**. Observando os sistemas interactivos mono-utilizador de utilização massiva, pode-se

afirmar com segurança, que o modelo apresentado apresenta uma maior riqueza semântica e uma especificação adequada para sistemas multi-utilizador.

A utilização do modelo no desenvolvimento da aplicação apresentada realçou algumas características de suporte ao desenvolvimento. **É fácil enriquecer o modelo com novas primitivas enquadrando-as nas classes definidas**. **Os mecanismos de estruturação de dados fornecidos apresentam-se como adequados a um conjunto vastíssimo de aplicações**. Note-se que com um conjunto mínimo de alterações sobre as representações dos Contentores e apresentações dos Conteúdos do *IDEAGEN*, seria possível construir um conjunto simples de aplicações cooperativas como editores de texto (representando a hierarquia de parágrafos, secções, etc.), gestores de ficheiros (representando as hierarquias de pastas e ficheiros), etc.

Outro conjunto de conclusões que puderam ser tiradas durante o desenvolvimento da aplicação prendem-se com o processo de desenvolvimento. **A utilização da metodologia DASCo em algumas fases de desenvolvimento, realçou a importância da utilização dos produtos assistentes e das técnicas de análise por separação de aspectos. Por outro lado, confirmou a necessidade de adaptação dos mecanismos tradicionais de controlo da concorrência para os requisitos de não transparência dos sistemas interactivos**.

De um ponto de vista mais global, importa analisar o enquadramento das soluções propostas e identificar tópicos que não foram analisados de forma tão profunda quanto seria desejável, ou que não o foram de todo. Considera-se que o seu estudo poderá e deverá ser objecto de trabalho futuro. Neste contexto apontam-se as necessidades sentidas na realização do trabalho.

O mecanismos de suporte e controlo da interacção foram estudados de forma independente das aplicação e focaram essencialmente os aspectos de interacção directa com o utilizador. Importa estudar a integração e adaptabilidade do modelo descrito a processos de grupo, como sejam por exemplo a facilitação de processos de decisão, ou de técnicas exploratórias de colaboração.

Dada a limitação do modelo à construção de estruturas de dados hierárquicas, importa estendê-lo para estruturas do tipo hipertexto de forma a suportar um conjunto ainda mais abrangente de aplicações. Acredita-se que essa extensão é simples, uma vez que não implica a introdução de novas primitivas de interacção.

Um estudo mais cuidado das primitivas de interacção permitirá encontrar técnicas gerais de recuperação, permitindo a inclusão dos mecanismos de *undo/redo*. A introdução destes mecanismos em interfaces multi-utilizador introduz um conjunto novo de problemas que deve ser tido em conta.

Durante o teste das técnicas de interacção apresentadas, identificaram-se alguns padrões de sequência de operações para diferentes utilizadores. Seria interessante desenvolver agentes assistentes de interface capazes de identificá-los e criar formas de automatizá-los.

O estudo das técnicas de concorrência não contemplou os aspectos de controlo de acesso. Admitiu-se que a manipulação de todos os objectos públicos é livre a todos os utilizadores. Na maioria das aplicações o acesso aos dados é restrito a classes de utilizadores. Este controlo poderá ser feito a partir dos mecanismos apresentados para controlo da concorrência.

Na apresentação das opções de desenho para a componente de interface, foram avançadas algumas ideias especulativas que não foram testadas. Seria importante poder experimentá-las de modo a estabelecer um conjunto de princípios de desenho em função do tipo de aplicação em causa. Note-se também, que o conjunto de opções indicado não é de modo algum fechado, convidando-se desde já os interessados, a "darem asas" ao seu espírito criativo, no sentido de encontrarem soluções inovadoras.

A descrição do modelo de suporte considerou apenas as classes de objectos Monitores que directamente interagem com os restantes objectos do modelo. No entanto, importa estudar outros mecanismos que permitam obter um conjunto de informações ainda mais vasto das actividades do grupo. Incluem-se os

aspectos de identificação dos membros do grupo, dos responsáveis pela realização das operações interactivas, controlo de acesso e outros que possam ser aplicáveis em contextos particulares.

No que diz respeito ao desenvolvimento de aplicações seria importante aplicar a metodologia DASCo a todas as fases respeitantes à componente de partilha de dados e estudar a influência de cada um dos aspectos nas componentes de estruturação da interacção e na interface utilizador seguindo o princípio da não transparência. Acredita-se que um estudo cuidado de cada um destes aspectos poderia conduzir à extensão do modelo, introduzindo novas componentes nos objectos de suporte, particularmente a criação de novas classes de Monitores.

BIBLIOGRAFIA

[Antunes 94]
P. Antunes e N. Guimarães. **Multiuser Interface Design in CSCW Systems**. *ESPRIT Basic Research Project 6360, Broadcast.*. Voume 3, Systems Engineering, Chapter 4. Outubro de 1994.

[Antunes 95a]
P. Antunes e N. Guimarães. **Structuring Elements for Group Interaction**. *Second Conference on Concurrent Engineering, Research and Applications (CE'95).* Washington D.C. Agosto de 1995.

[Antunes 95b]
P. Antunes, N. Guimarães, J. Segovia e J. Cardenosa. **Beyond Formal Processes: Augmenting Workflow with Group Interaction Techniques**. *Conference on Organizational Computing Systems (COOCS'95).* Agosto de 1995.

[Antunes 95c]
P. Antunes e N. Guimarães. **NGTool - Exploring Mechanisms of Support to Interactivity in the Group Process**. *CYTED-RITOS International Workshop on Groupware.*. Lisboa. Setembro de 1995.

[Antunes 96a]
P. Antunes e N. Guimarães. **User-Interface Support to Group Interaction**, Em: *Second CYTED-RITOS International Workshop on Groupware CRIWG'96*. CYTED-RITOS, Puerto Varas, Chile. Setembro de 1996.

[Antunes 96b]
P. Antunes. **Organizações, Grupos e Tecnologia: Suporte Computacional a Processos de Interacção e Decisão em Grupo**. *Dissertação para obtenção do grau de doutor em Engenharia Electrotécnica e de Computadores.* Instituto Superior Técnico, Universidade Técnica de Lisboa. Novembro de 1996.

[Balasubramanian 93]
V. Balasubramanian. **Hypermedia Issues and Applications: A State-of-the-Art Review**. *World Wide Web*. (*http://cbl.leeds.ac.uk/nikos/tmp/hypemedia/hypemedia.html*). Dezembro de 1993.

[Barghouti 91]
N. Barghouti and G. Kaiser. **Concurrency Control in Advanced Database Systems**. *ACM Computing Surveys*. Setembro de 1993.

[Brockschmidt 94]
K. Brockschmidt. **Inside OLE2**. *Microsoft Press.* 1994.

[Cornell 96]
G. Cornell e C. Horstmann. **Core Java**. *Sun Microsystems Inc.* 1996.

[Ellis 91]
C. Ellis, S. Gibbs e G. Rein. **Groupware: Some Issues and Experiences**. *Communications of the ACM*. 1991.

[Gamma 95]
E. Gamma, R. Helm, R. Johnson e J. Vlissides. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison Wesley. 1995.

[Gil 96]
L. Gil e J. Martins. **Implementação em C++ de uma *framework* para concorrência**. *Relatório do Trabalho Final de Curso*. Lic. em Engª Informática e de Computadores, Instituto Superior Técnico, Universidade Técnica de Lisboa. Setembro de 1996.

[Gray 93]
J. Gray e A. Reuter. **Transaction Processing: Concepts and Techniques**. Morgan Kaufmann. 1993.

[Greenberg 94]
S. Greenberg e D. Marwood. **Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface**. *ACM 1994 Conference on Computer Supported Cooperative Work CSCW '94*. Chapel Hill, North Carolina. Outubro de 1994.

[Greif 88]
I. Greif e S. Sarin. **Data Sharing in Group Work**. *Computer-Supported Cooperative Work: a Book of Readings*. Morgan Kaufmann Publishers Inc. 1988.

[Ho 96]
T. Ho e I. Soares. **Ferramentas de Trabalho Cooperativo**. *Relatório do Trabalho Final de Curso*. Lic. em Engª Informática e de Computadores. Instituto Superior Técnico. Julho de 1996.

[Johnson 94]
P. Johnson. **Experiences with EGRET: An Exploratory Group Work Environment**. *Collaborative Computing*. 1994.

[Kaplan 92]
S. Kaplan, W. Tolone, D. Bogia e C. Bignoli. **Flexible, Active Support for Collaborative Work with Conversation Builder**. *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work.*. 1992.

[Marques 90]
J. A. Marques e P. Guedes. **Fundamentos de Sistemas Operativos**. Editorial Presença. 1990.

[Paulo 91]
V. Paulo. **EdGar – Ferramenta Interactiva de Criação e Manipulação de Grafos**, *Relatório do Trabalho Final de Curso*. Instituto Superior Técnico. Universidade Técnica de Lisboa. Novembro de 1996.

[Pressman 91]
R. Pressman. **Software Engineering - A Practitioner's Approach**, *3ª Edição*. McGraw-Hill International. 1991.

[Rational 97]
Rational Software Corporation. **Unified Modeling Language v 1.1c - Notation Guide**, *World Wide Web (http://www.rational.com)*. Julho de 1997.

[Rodden 92]
T. Rodden, J. Mariani, e G. Blair. **Supporting Cooperative Applications**, *Computer Supported Cooperative Work*. Kluwer Academic Publishers. 1992.

[Schmidt 94]
D. Schmidt. **The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software**, *11th and 12th Sun User Group Conferences*. San Jose, California. Dezembro de 1993 e Junho de 1994.

[Schmidt 95]
D. Schmidt. **The ACE Object-Oriented Encapsulation of Lightweight Concurrency Mechanisms**, *Technical Report WUCS-95-31*. Washington University. St. Louis. 1995.

[Silva 95]
A. R. Silva, P. Sousa e J. A. Marques. **Development of Distributed Applications with Separation of Concerns**, *Proceedings of the 1995 Asia-Pacific Software Engineering Conference APSEC' 95*. Brisbane, Austrália. Dezembro de 1995.

[Silva 96a]
A. R. Silva, J. Pereira e J. A. Marques. **Customizable Object Synchronization Pattern**.*European Conference on Pattern Languages of Programs - EuroPlop'96*. Kloster Irsee, Alemanha. Julho de 1996.

[Silva 96b]
A. R. Silva, L. Gil e J. Martins. **Three-Layered Framework with Separation of Concerns**, *OOPSLA'96 Workshop on Exploration of Framework Design Principles*. San Jose, California. Outubro de 1996.

[Silva 97a]
A. R. Silva, J. Pereira e J. A. Marques. **Customizable Object Recovery Pattern**. In *Pattern Languages of Program Design 3 Book*. Addison-Wesley. 1997. (A Sair)

[Silva 97b]
A. R. Silva. **Framework, Design Patterns and Pattern Language for Object Concurrency**. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*. Las Vegas, Nevada, EUA. Junho/Julho de 1997.

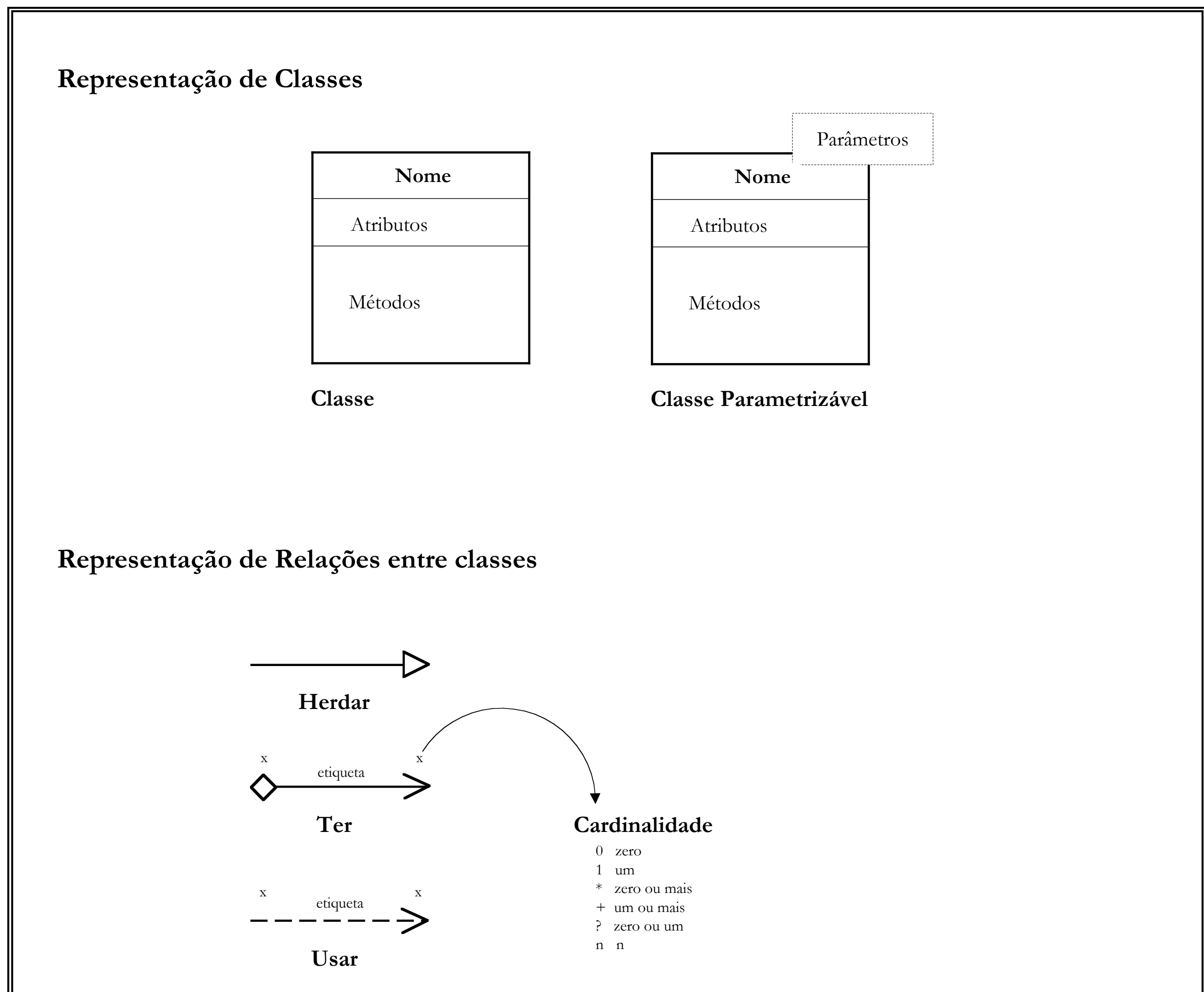
[Silva 97c]
A. R. Silva. **A Quality Design Solution for Object Synchronization**. *Proceedings of the European Conference on Parallel Processing*. Passau, Alemanha. Agosto de 1997.

[Silva 98]
A. R. Silva. **Development and Extension of Frameworks**. In *Handbook of Object Technology*. Saba Zamir Editor. CRC Press. 1998. (A Sair)

Apêndice A

RESUMO DA NOTAÇÃO UML

Apresenta-se o significado resumido do subconjunto utilizado da notação UML. A semântica desta notação é muito mais vasta, para um conhecimento aprofundado sugere-se a consulta de [Rational 97].



Apêndice B

EXEMPLOS DE UTILIZAÇÃO

Apresenta-se um conjunto de figuras que ilustra a realização de um conjunto de operações exemplificativo do modelo de interacção. Os exemplos foram gerados no IDEAGEN.

B.1 Interacções disponibilizadas a um utilizador

As figuras apresentadas neste tópico apresentam cada uma duas imagens, que apresentam o estado visual da aplicação antes e depois da realização de uma Conexão. As setas representadas na primeira imagem representam o movimento de *drag'n'drop* realizado. Todas as interacções são mostradas no espaço privado. No entanto, o seu efeito seria o mesmo no espaço público.

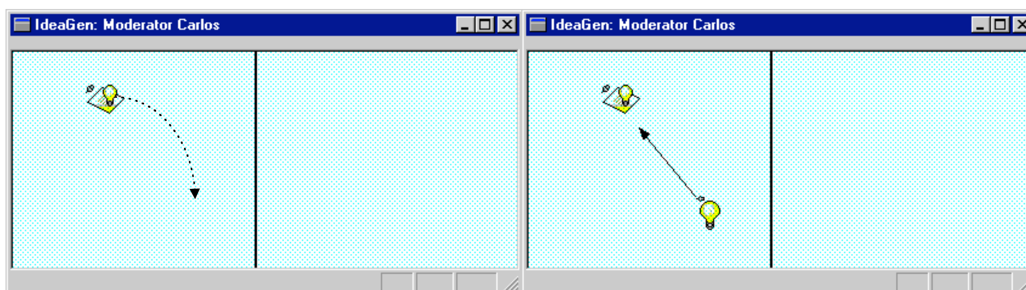


Figura 42- Conexão entre uma Origem durável e o espaço livre (I)

A Origem gera um novo objecto. A pista dada pelo espaço livre induz o Contentor Origem como Destino.

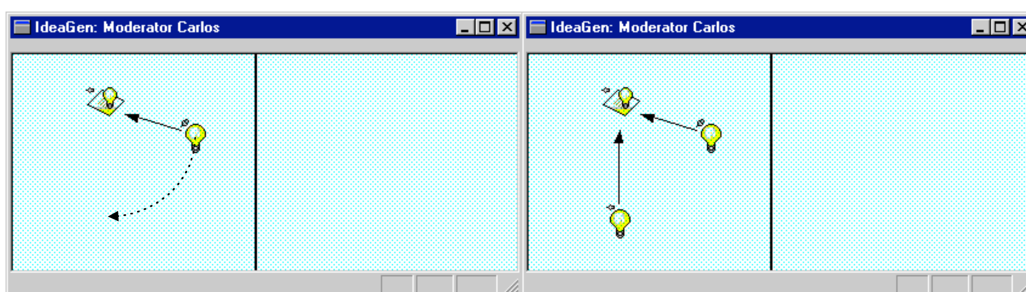


Figura 43 - Conexão entre uma Origem durável e o espaço livre (II)

A Origem gera um novo objecto à sua imagem. A pista dada pelo espaço livre induz o Contentor onde se encontra a Origem.

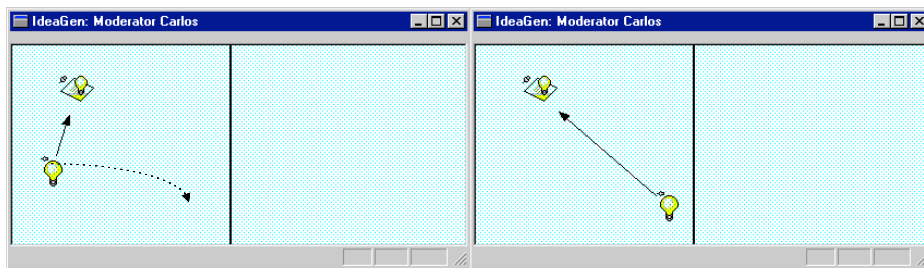


Figura 44 - Conexão entre uma Origem transitória e o espaço livre (I)

A Transferência é a Origem. A pista dada pelo espaço livre induz o Contentor onde se encontra a origem como Destino.

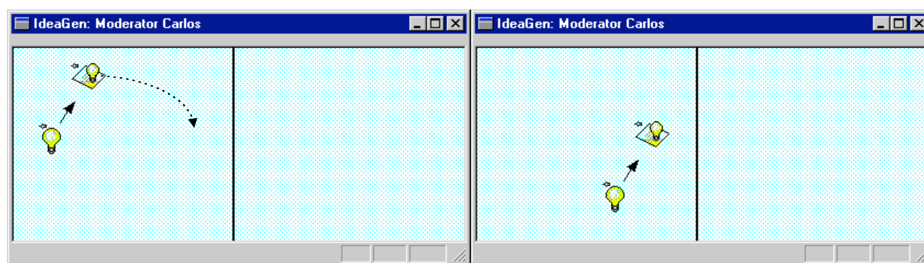


Figura 45 - Conexão entre uma Origem transitória e o espaço livre (II)

A Transferência é a Origem. A pista dada pelo espaço livre induz o Espaço Privado como Destino. A Origem mantém-se no mesmo ponto da estrutura de dados mas é mudada para outro local apenas em termos visuais.

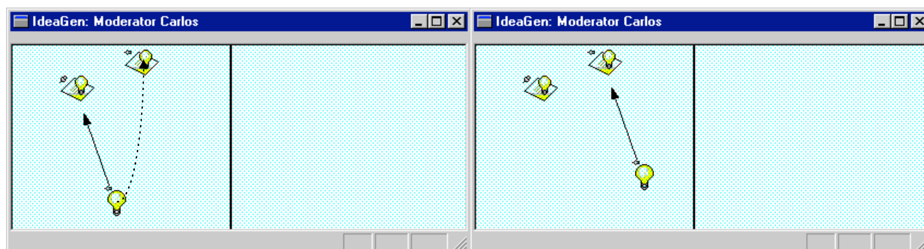


Figura 46 - Conexão entre uma Origem transitória e outro Contentor (I)

A Transferência é a origem. O Destino é claramente identificado. A Origem passa para o Conteúdo composto do Destino.

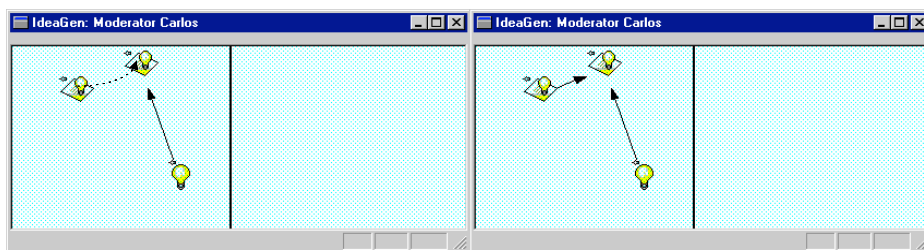


Figura 47 - Conexão entre uma Origem transitória e outro Contentor (II)

O resultado é idêntico ao do exemplo anterior, exemplificando-se a criação de estruturas hierárquicas. Um Contentor de Conteúdo composto é colocado no interior de outro com as mesmas características.

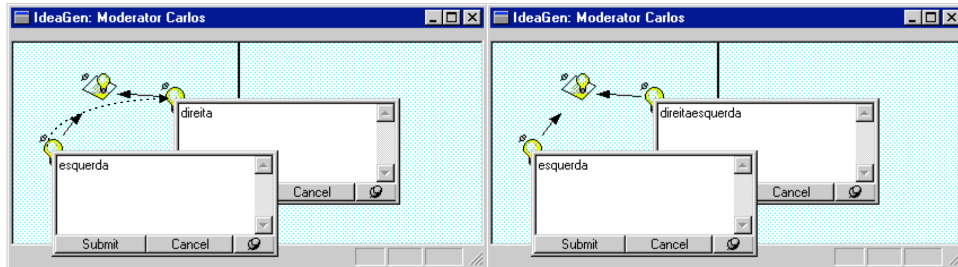


Figura 48 - Conexão entre um Contentor durável de Conteúdo simples durável e outro Contentor (I)

O Conteúdo da Origem mantém-se por ser durável. O Conteúdo do Destino (por ser durável) passa a conter o seu valor inicial junto com o Conteúdo da Transferência.

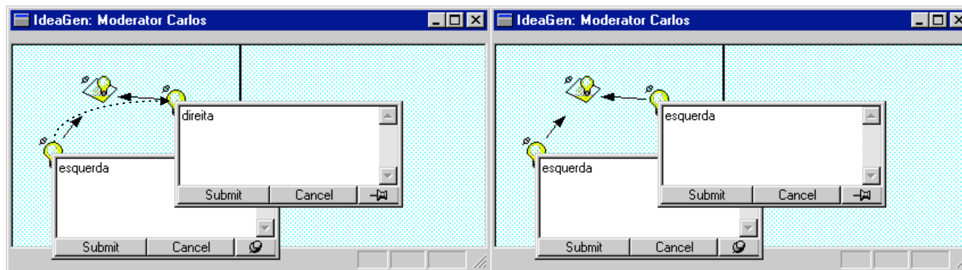


Figura 49 - Conexão entre um Contentor durável de Conteúdo simples durável e outro Contentor (II)

O Conteúdo da Origem mantém-se por ser durável. O Conteúdo do Destino (por ser transitório) é substituído pelo do Conteúdo da Transferência.

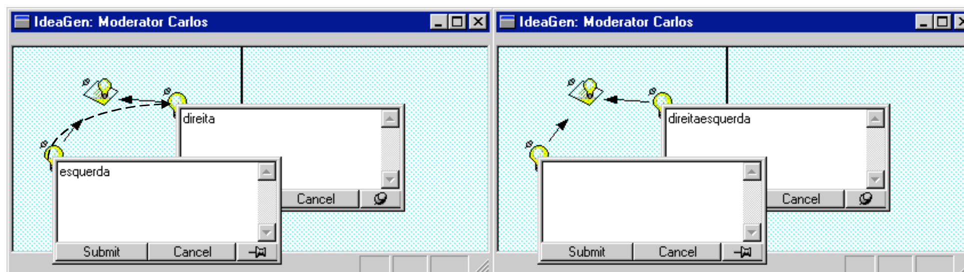


Figura 50 - Conexão entre um Contentor durável de Conteúdo simples transitório e outro Contentor (I)

O Conteúdo da Origem é eliminado por ser transitório. O Conteúdo do Destino (por ser durável) passa a conter o seu valor inicial junto com o Conteúdo da Transferência.

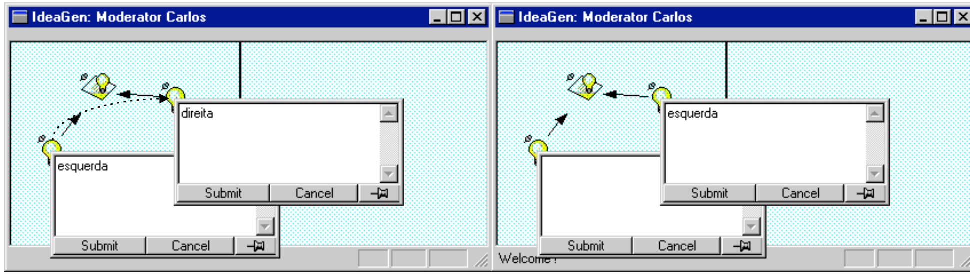


Figura 51 - Conexão entre um Contentor durável de Conteúdo simples transitório e outro Contentor (II)

O Conteúdo da Origem é eliminado por ser transitório. O Conteúdo do Destino (por ser transitório) é substituído pelo do Conteúdo da Transferência.

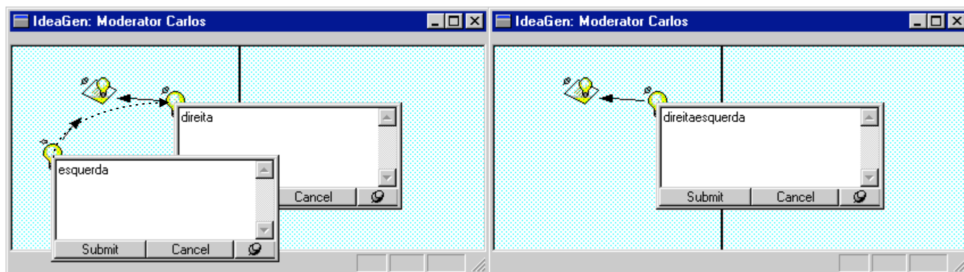


Figura 52 - Conexão entre um Contentor transitório de Conteúdo simples transitório e outro Contentor (I)

A Origem é eliminada por ser transitória. O Conteúdo do Destino (por ser durável) passa a conter o seu valor inicial junto com o Conteúdo da Transferência. A propriedade de manipulação dos Conteúdos é absorvida pela dos Contentores.

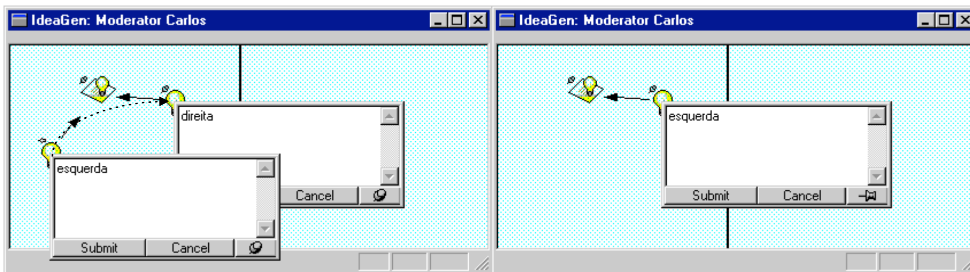


Figura 53 - Conexão entre um Contentor transitório de Conteúdo simples transitório e outro Contentor (II)

A Origem é eliminada por ser transitória. O Conteúdo do Destino (por ser transitório) é substituído pelo do Conteúdo da Transferência.

B.2 Distribuição e relaxação da coerência visual

As figuras apresentadas neste tópico apresentam cada uma quatro imagens, que apresentam o estado visual de dois nós da aplicação antes e depois da realização de uma Conexão. As duas imagens superiores apresentam a situação antes da Conexão e as inferiores o resultado final. As setas representadas na primeira imagem representam o movimento de *drag'n'drop* realizado. Todas as interacções são mostradas no espaço público. No espaço privado não afectam os restantes nós da aplicação.

Pretende-se apresentar os mecanismos de distribuição das operações interactivas e as técnicas de relaxação da coerência visual. A utilização ou não destas técnicas é deixada em aberto pelo modelo. Apresentam-se aqui as adoptadas para o *IDEAGEN*, sendo estas dependentes da aplicação. Os autores recomendam a utilização deste tipo de relaxação de modo a promover o trabalho em paralelo e não introduzir elementos distractivos na interface. Todas as operações não visualizadas por razões de incoerência são monitorizadas.

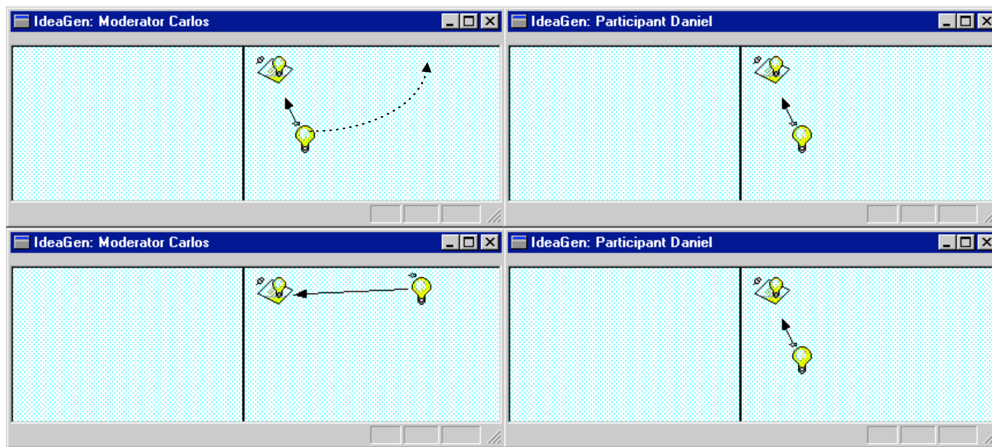


Figura 54 - Conexão que não altera a estrutura hierárquica dos Contentores

O Contentor Origem e Destino são o mesmo. A Transferência não é alterada. Nestes casos a semântica da Conexão é apenas organizar o espaço visual sem afectar a estruturação dos dados. A coerência visual pode ser relaxada.

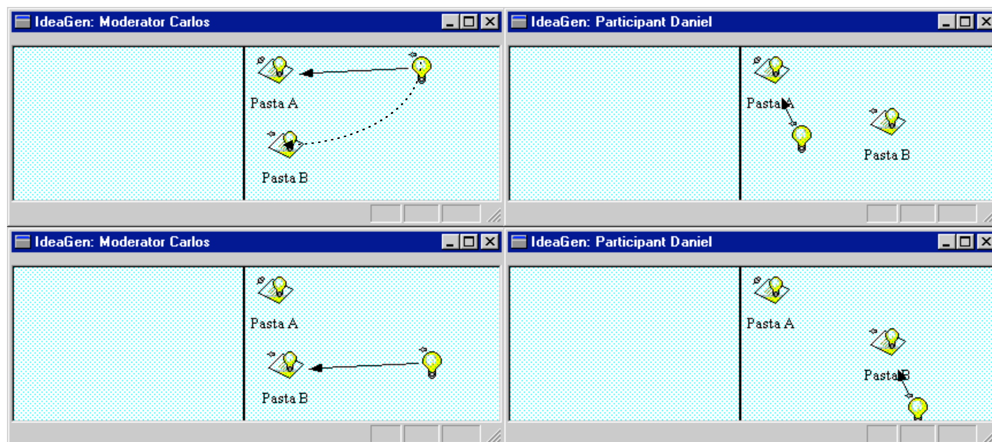


Figura 55 - Conexão que altera a estrutura hierárquica dos Contentores

As vistas mantém-se incoerentes e é realizada uma Conexão que altera a estrutura. Distribui-se a Conexão e as alterações visuais são feitas em função dos estados particulares dos participantes.

Sequência para monitorização de alterações.

No primeiro passo o utilizador edita um Conteúdo simples.

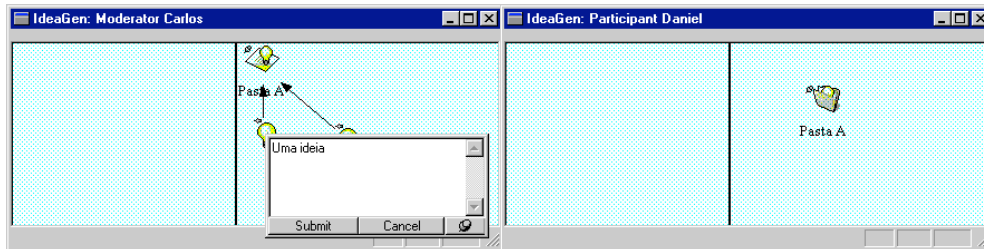


Figura 56 – Monitorização da edição de Conteúdo simples (a)

Ao submeter a sua edição, os utilizadores que não mantém a apresentação do Conteúdo aberta são avisados da edição pelo monitor de gestão de interface.

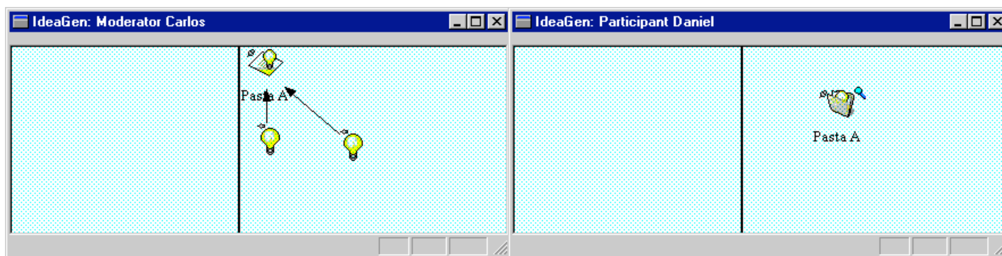


Figura 57 - Monitorização da edição de Conteúdo simples (b)

O utilizador pode consultar a alteração seguindo os monitores de gestão de interface.

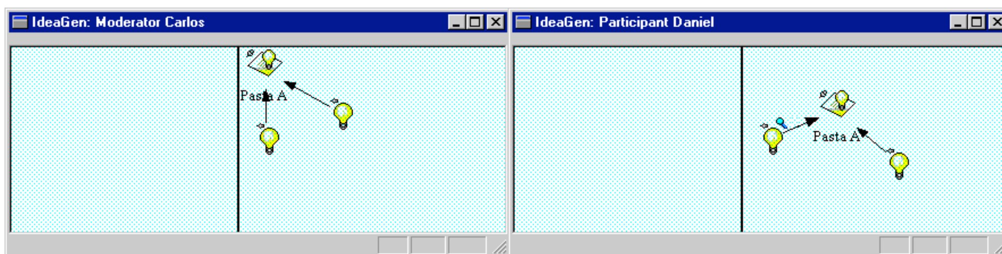


Figura 58 - Monitorização da edição de Conteúdo simples (c)

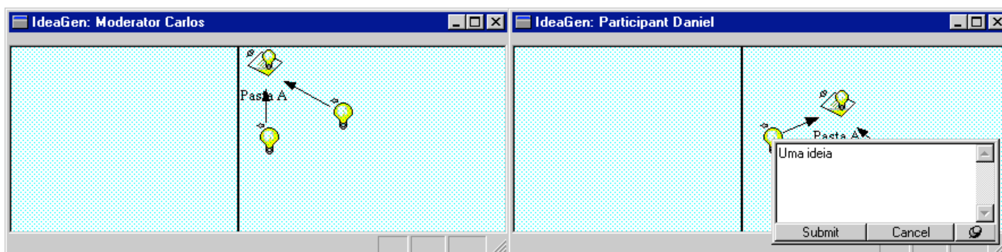


Figura 59 - Monitorização da edição de Conteúdo simples (d)

B.3 Controlo da concorrência

Nesta secção apresentam-se dois exemplos ilustrativos dos mecanismos de controlo apresentados no Capítulo 5.

Sequência de edições concorrentes.

O primeiro utilizador inicia a edição do Conteúdo simples.

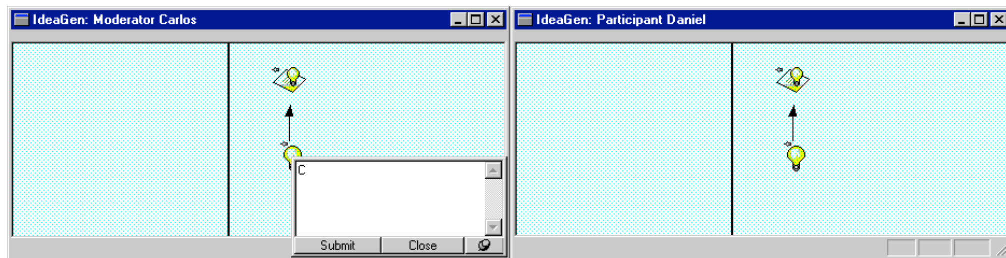


Figura 60 - Edição de dois Conteúdos simples em simultâneo (a)

O segundo utilizador inicia também a edição do mesmo Conteúdo. O conflito é monitorizado pelo semáforo amarelo.

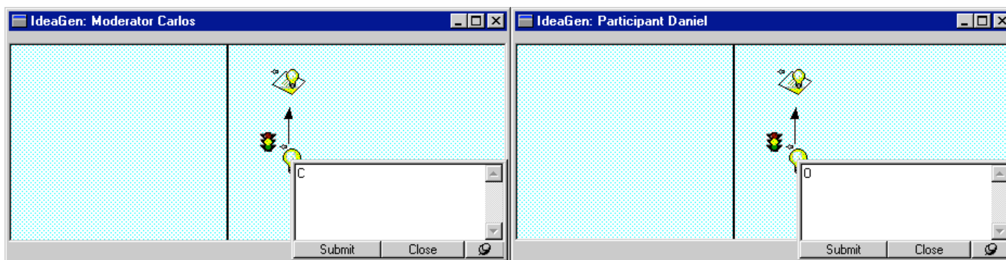


Figura 61 - Edição de dois Conteúdos simples em simultâneo (b)

O primeiro utilizador submete o Conteúdo. Deixa de poder editar e o semáforo passa a vermelho.

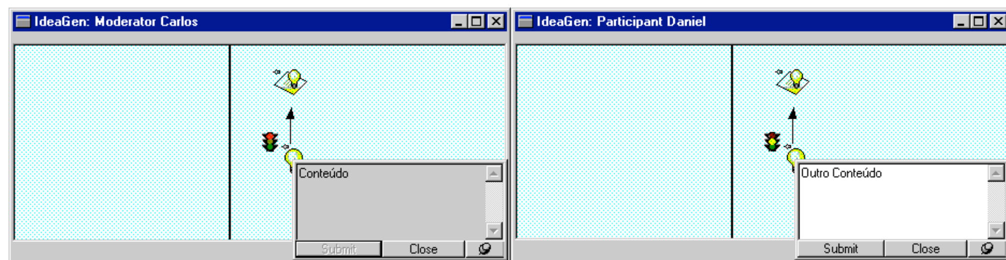


Figura 62 - Edição de dois Conteúdos simples em simultâneo (c)

O segundo utilizador submeteu também o Conteúdo. Ninguém se encontra a editar pelo que se inicia escolha de versão. Ambos os utilizadores votam uma versão. O mais votado repõe a coerência de versões. O semáforo passa a verde por momentos, desaparecendo depois.

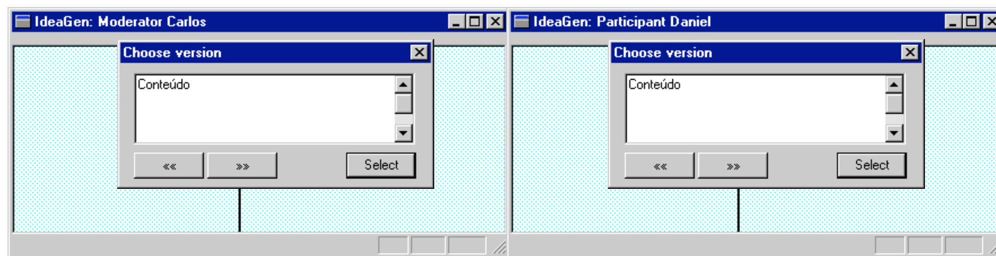


Figura 63 - Edição de dois Conteúdos simples em simultâneo (d)

Sequência de manipulação concorrente de Contentores.

O primeiro utilizador inicia uma Conexão.

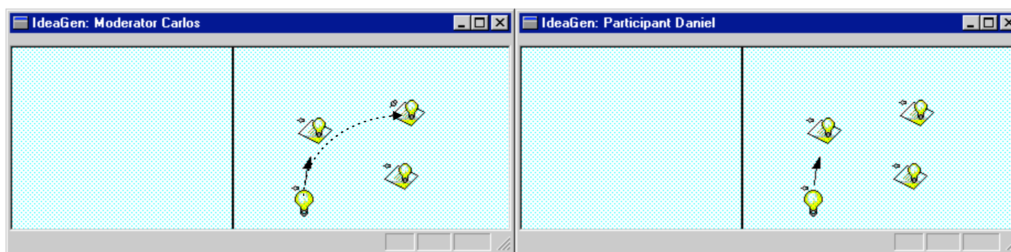


Figura 64 – Manipulação de Contentores (a)

O segundo tenta iniciar a Conexão no mesmo Contentor, sendo avisado da sua impossibilidade pelo semáforo vermelho. Contínua a interacção sendo o seu efeito nulo.

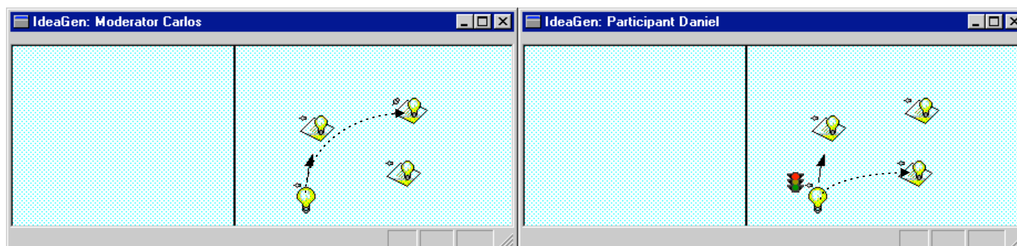


Figura 65 – Manipulação de Contentores (b)

Ambos os utilizadores terminam a Conexão. Só a do primeiro é efectivada.

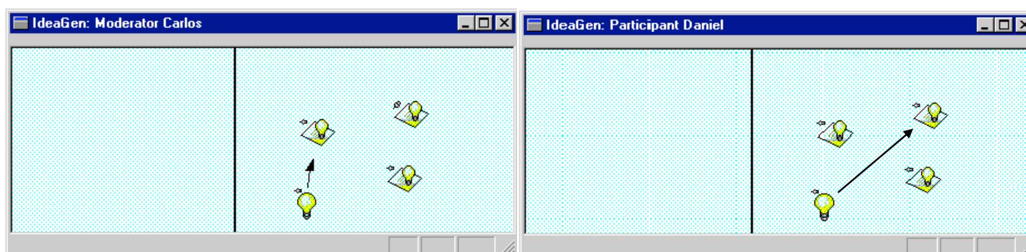


Figura 66 – Manipulação de Contentores (c)

Apêndice C

IMPLEMENTAÇÃO EM C^{++}

Neste apêndice apresentam-se excertos da implementação em C^{++} dos modelos de objectos apresentados ao longo do texto.

C.1 Suporte à interacção

```
/*------------------------------------------------------------
 |
 |   class Container
 |
 |   An object that has an information Content and can be
 |   manipulated for information structuring.
 |
 *----------------------------------------------------------*/

class Container {

public:

   Container (Container *_referenced_by = NULL,
             int _visibility = PRIVATE,
             int _manipulation = DURABLE,
             Content *_content = NULL);

   virtual ~Container ();

// setting data
   void  setVisibilityProp(int _value)  { visibility = _value; }
   void  setManipulationProp(int _value){ manipulation =
                                          _value; }
   void  toggleManipulationProp()
        { manipulation =
             ((manipulation == DURABLE) ? TRANSIENT : DURABLE);
        }
   void  setContent(Content *_content)  { content = _content; }
   void  setReferencedBy(Container *_referenced_by)
        { referenced_by = _referenced_by; }

// interaction
   // Verifies if the container is (in)directly referenced by
   // the argument container
   int isIndirectlyReferenced (Container *indirect_reference);
   // Returns a known container that can support others
   // containers
   Container *getSupportContainer ();
   // Changes the origin and defines the Transfer(return value)
   virtual Container   *preLinkAsOrigin();
   // Changes the destiny given the Transfer returned by the
origin
   virtual void         postLinkAsDestiny(Container *transfer);
   // Dereferenciates form "referenced_by"
   void  dereference();

// data accessing
   int   getVisibilityProp() { return visibility; }
   int   isPublic()          { return (visibility == PUBLIC); }
   int   isPrivate()         { return (visibility == PRIVATE); }
   int   getManipulationProp() { return manipulation; }
   int   isTransient()   { return (manipulation == TRANSIENT); }
   int   isDurable()     { return (manipulation == DURABLE); }
   Container   *getReferencedBy()  { return referenced_by; }
   Content     *getContent()       { return content; }

   // sets the visibility property for all member containers
   //(if composed)
   void  setVisibilityPropOnCascade(int _value);
   // some applications may need to distinguish between
   //containers
   virtual int getType()    { return UNDEFINED_TYPE; }

   // (de)referentiates a Container form the content (if
   // composed)
   void  referenceFromContent(Container *for_adding);
   void  dereferenceFromContent(Container *for_dereference);

// graphical representation methods

   // container representation
   virtual void openView()         { view_state = VISIBLE; }
   virtual void closeView()        { view_state = HIDDEN; }
   virtual int isOpen()     { return (view_state == VISIBLE); }
   virtual int isClosed()   { return (view_state == HIDDEN); }

   // content representation
   virtual void openContentView();
   virtual void closeContentView();
   virtual int contentIsOpen();
   virtual int contentIsClosed();

// pre-defined values

   enum {PUBLIC, PRIVATE};
   enum {TRANSIENT, DURABLE};
   enum {VISIBLE, HIDDEN};
   enum {UNDEFINED_TYPE = -1};

protected:

// operations
   // substitutes all the references for the Container with
   // references for another one
   void  substitutes(Container *substitute);

   // returns a new Container when the Container can generate
   // others (is durable)
   virtual Container *generate() { return NULL; }

// data
   // dynamic properties
   int   manipulation;
   int   visibility;

   // known objects
   Content     *content;
   Container   *referenced_by;

   // graphical state
   int   view_state;
};
```

Figura 67 - Classe Container

```
/*--------------------------------------------------------------
 |
 |   preLinkAsOrigin and postLinkAsDestiny
 |
 |   Methods that implement the interaction patterns defined by
 |   the user Connections.
 |
 *-------------------------------------------------------------*/

Container *Container::preLinkAsOrigin()
{
   // The container to be transfered
   Container *transfer;

   // for the Container
   // each Container has is generation policy
   if(isDurable())
      transfer = generate();
   else // isTransient()
   {
      dereference();
      transfer = this;
   }

   // for the Content
   if(content)
      content->containerPreLinked();

   if(content && content->isTransient())
      content->null();

   return transfer;
}


void Container::postLinkAsDestiny(Container *transfer)
{
   // a ComposedContent
   if(content && content->isSimple())
      content->containerPostLinked(transfer->getContent());

   if(content && content->isComposed())
      referenceFromContent(transfer);

   transfer->setReferencedBy(this);

   transfer->setVisibilityPropOnCascade(visibility);
}
```

Figura 68 – Métodos de iniciação como Origem e finalização como Destino

```
class Content {

public:

   Content (int _manipulation = DURABLE,
            Container *_referenced_by = NULL );

   virtual ~Content ();

// setting data
   void  setManipulationProp(int _value)
                           { manipulation = _value; }
   void  toggleManipulationProp()
            { manipulation = ((manipulation == DURABLE) ?
                        TRANSIENT : DURABLE);}
   void  setReferencedBy(Container *_referenced_by)
                        { referenced_by = _referenced_by; }
   void  dereference()
                        { referenced_by->setContent(NULL); }

   virtual void  add(Content *data_to_add) = 0;
   virtual void  reset(Content *new_data)  = 0;
   virtual void  null()  = 0;

// interaction
   // Implements the interaction patterns
   virtual void  containerPreLinked();
   virtual void  containerPostLinked
                 (Content *transfer_content);
   // cloning a Content for data transfer in connections
   virtual Content *clone() = 0;

// obtaining data
   int   getManipulationProp()   { return manipulation; }
   int   isTransient()
         { return (manipulation == TRANSIENT); }
   int   isDurable()
         { return (manipulation == DURABLE); }
   virtual int   isSimple()   = 0;     // simple content
   virtual int   isComposed() = 0;     // composed content
                              // "Container of Container's"
   Container *getReferencedBy()   { return referenced_by; }

// graphical representation of a content
   virtual void openView()    { view_state = VISIBLE; }
   virtual void closeView();
   int isOpen()    { return (view_state == VISIBLE); }
   int isClosed()  { return (view_state == HIDDEN); }

// pre-defined values
   enum {TRANSIENT, DURABLE};
   enum {VISIBLE, HIDDEN};

protected:

// data

   // dynamic properties
   int   manipulation;

   // known objects
   Container   *referenced_by;

   // graphical state
   int   view_state;
};
```

Figura 69 - Classe Content

```
class Connection {

public:

   Connection ();

   ~Connection () {};

// Connection phases
   virtual void start(Container *_origin);
   virtual int finish(Container *_destiny_hint);
         // returns the success of the
         // connection operation

// At application level, some connections may not be
// available
// Potentialy all connections are accepted by the model
   virtual int connectionAllowed()   { return SUCCESS; }

// At application level, the destiny may be different from
// the destiny_hint
   virtual Container *getDestiny() { return destiny_hint; }

// Finish return values

   enum {FAIL = 0, SUCCESS};

protected:

// Containers envolved
   Container *origin;
   Container *destiny_hint;
   Container *destiny;
   Container *transfer;
};

void Connection::start(Container *_origin)
{
   origin = _origin;
}

int Connection::finish(Container *_destiny_hint)
{
   destiny_hint = _destiny_hint;

   // determines whether the transfer is allowed or not
   if(!connectionAllowed())
      return Connection::FAIL;

   // THE CONNECTION IS ALLOWED !! :-)

   // the transfer is created
   transfer = origin->preLinkAsOrigin();

   // destiny calculation, given the origin and the
   // destiny hint
   destiny = getDestiny();

   // the transfer is delivered to the destiny
   destiny->postLinkAsDestiny(transfer);

   // the connection was made with success
   return Connection::SUCCESS;
}
```

Figura 70 - Classe Connection e seus métodos de iniciação e finalização

C.2 Controlo da interacção

```
class SyncRecSCon :
            public SimpleContent,
            public Opaque_Passive_Sync_Rec_Inter<RecSContent>
{

public:

  SyncRecSCon(ComposedContent *parent, UniqID id):
    SimpleContainer(parent, id),
    Opaque_Passive_Sync_Rec_Inter<RecSContent>
                (new RecSContent(parent, id),
                 new Generic_Obj_Synchronizer(),
                 new DefUpd_Obj_Recoverer<RecSContent>()),
    sync_data(new SContentSyncData(&meeting_users)),
    real_content(new RealSimpleContent(parent, id))
    {}

  ~SyncRecSCon()
    { delete sync_data; }


  void startEdit(User *actor);
  void endEdit(User *actor, char *submission);
  void chooseVersion(User *actor, int selection_id);

  // user interface non transparency

  void broadcastSolutions(Users *actors);
  void broadcastConflict(Users *actors);
  void broadcastFree(Users *actors);
  void setUneditable(User *actor);

private:

  Passive_Sync_Rec_Inv<RecSContent> *
    getPendingInteraction(User *actor,
                  ContentInvocation *ret_inv);

  SContainerSyncData *sync_data;
  MutualExQueue<ContentInvocation> pending_invocations;

  RealSimpleContent *real_content;
};


/*-----------------------------------------------------------*/

void SyncRecSCon::startEdit(User *actor)
{
  Passive_Sync_Rec_Inv<RecSContent> *invocation;
  int state_before = sync_data->getState();

  invocation = createInvocation
       (new EditPred<Passive_Sync_Rec_Inv<RecSContent> >
                      (sync_data),
        new CopySContentRecPoint());

  pending_invocations.insert
     (new ContentInvocation(actor, invocation));

  if(SimpleContent *content =
     (SimpleContent *)preControl(invocation))
    content->startEdit(actor);

  if(state_before == SContentSyncData::FREE &&
     sync_data->getState() == SContentSyncData::ONE_EDITOR)
    broadcastConflict(sync_data->getActors());
}

/*-----------------------------------------------------------*/

void SyncRecSCon::endEdit(User* actor, char *submission)
{
  Passive_Sync_Rec_Inv<RecSContent> *invocation;
  ContentInvocation *ret_inv;

  int state_before = sync_data->getState();

  if(invocation = getPendingInteraction(actor, ret_inv))
  {
    if(!postControl(invocation))
      endEdit(actor, submission);
    pending_invocations.remove(ret_inv);
  }

  if(state_before == SContentSyncData::ONE_EDITOR &&
     sync_data->getState() == SContentSyncData::FREE)
  {
    real_content->setContent(submission);
    versions.deleteAll();
    broadcastFree(sync_data->getActors());
  }

  if(state_before == SContainerSyncData::CONFLICT &&
     sync_data->getState() == SContainerSyncData::SOLVING)
    broadcastSolutions(sync_data->getActors());

  if(state_before == SContainerSyncData::CONFLICT)
    setUneditable(actor);
}


/*-----------------------------------------------------------*/

void SyncRecSCon::chooseVersion(User *actor, int selection_id)
{
  int state_before = sync_data->getState();

  int counter;
  GPListCursor(Version) cursor(&versions);
  Version *the_version;
  Version *current;
  int max_votes = 0;

  for(cursor.goHead(), counter = 1;
      counter < selection_id;
      cursor.goForth(), counter++);

  the_version = (Version *)cursor.value();

  the_version->incVotes();

  sync_data->decSolvers();

  if(sync_data->getSolvers() == 0)
  {
    // get the most voted one
    for(cursor.goHead(); cursor.isInList(); cursor.goForth())
    {
      current = (Version *)cursor.value();
      if(current->getVotes() > max_votes)
      {
         max_votes = current->getVotes();
         the_version = current;
      }
    }

    sync_data->setState(SContentSyncData::FREE);
    setContent(the_version->getContent());
    broadcastFree(sync_data->getActors());
    versions.deleteAll();
  }

}

/*-----------------------------------------------------------*/
```

Figura 71 - Controlo da Concorrência na edição de Conteúdos simples.

C.3 Instanciação no *IDEAGEN*

```
/*-----------------------------------------------------------
 |
 |   class EdgarNodeContainer
 |
 *-----------------------------------------------------------*/

class EdgarNodeContainer : public Container {

   public:

      EdgarNodeContainer (Container *referenced_by,
                      int visibility,
                      int manipulation);

      EdgarNodeContainer (Container *referenced_by,
                    int visibility, int manipulation,
                    NodeViewPtr _view, Point _node_position,
                    UniqID _node_id);

      virtual ~EdgarNodeContainer ();


      // graphical representation manipulation definition at
      // this application level
      virtual void openView   ();
      virtual void closeView  ();
      virtual void updateView ();
      virtual void openLabel  ();
      virtual void closeLabel ();
      virtual void updateLabel();

      // creating a view given a graphical and communication
      // environment
      // if the optional parameters are not used, the view is
      // created with
      // the actual graphical environment state.
      void createNode(Point new_position);
      void createLink(UniqID end);

      void deleteNode();
      void deleteLink();
      void updateLink(UniqID end);
      int  hasView()                     { return has_view; }

      // the identifier of the node bitmap representation
      virtual int getBitmap() = 0;

      // an help message about the container functionality
      virtual char *getMicroHelp() = 0;

      // the node position
      Point getPosition()          { return node_position; }
      Point getDisplayPosition()   { return display_position; }
      void  setPosition(Point _position);

      // returns the last node move (in a vector)
      Point movedBy()              { return last_move; }

      // returns the relative position of the node to
      // it's support container
      Point getReferentialPosition();

      // manipulation of node and link id's in the context of
      // the application graph
      void setLinkID(UniqID _link_id)    { link_id = _link_id; }
      UniqID getLinkID()                 { return link_id; }
      void setNodeID(UniqID _node_id)    { node_id = _node_id; }
      UniqID getNodeID()                 { return node_id; }

      NodeViewPtr getNodeView()    { return node_view; }

      GString *getLabel()
         { return ((ComposedContent *)content)->getLabel(); }
      void setLabel(GString *label)
         { ((ComposedContent *)content)->setLabel(label); }

      // broadcast node view container information
      virtual void broadcastInformation() {};
      virtual void broadcastContent    () {};
      virtual void broadcastStartEdit  () {};
      virtual void broadcastEndEdit    () {};

      virtual void redrawDialog        () {};

      virtual char *getNodeInfo() { return NULL; };

      // application graphical state
      static  BSTCallback *bst_cbk;

      virtual void setMsgID (int _msg_id) {};
      virtual int  getMsgID ()            { return -1; };
      virtual void setExpectedConfirmations
                  (int n_confirmations) {};
      virtual void receiveConfirmation ()  {};

      virtual void  activateInterfaceChangeMonitor () { };
      void    desactivateInterfaceChangeMonitor ()
           { is_interface_monitor_activated = FALSE; };
      virtual Point getInterfaceChangeMonitorViewPosition();

      void    activateConcurrencyMonitor    ()
        { is_concurrency_monitor_activated = TRUE;  };
      void    desactivateConcurrencyMonitor ()
        { is_concurrency_monitor_activated = FALSE; };
      int     isConcurrencyMonitorActivated ()
        { return is_concurrency_monitor_activated;  };
      virtual Point getConcurrencyMonitorViewPosition();

   protected:

      // indicates whether the view is created or not
      int has_view;

      NodeViewPtr node_view;
      LinkViewPtr link_view;
      UniqID      node_id;
      UniqID      link_id;
      Point       node_position;
      Point       last_move;
      Point       display_position;

      // for containers with composed content
      CLabelEdit *label_control;
      CFont      *font;
      BOOL        has_created_label;

      BOOL is_interface_monitor_activated;
      BOOL is_concurrency_monitor_activated;

      ConcurrencyMonitor     *concurrency_monitor;
      InterfaceChangeMonitor *interface_change_monitor;
};
```

Figura 72 – Instanciação de Container no *IDEAGEN*

```
/*-------------------------------------------------------------
 |
 | openView
 |
 | Opens the view of the node. In this case, an edgar node view
 | is created on top of a existing graph node.
 |
 *-----------------------------------------------------------*/
void EdgarNodeContainer::openView()
{
   // the labels are not from edgar
   char *label;

   // the node has a internal position that can be not valid for
   // some window or graph views parameters.
   // This display position is an auxiliary position used in
   // such cases.
   display_position = node_position;

   // invocation cannot be attended
   if(!hasView() || isOpen())
      return;

   // the content has a label
   if(content && content->isComposed() &&
      ((ComposedContent *)content)->getLabel())
   {
      label = new char[((ComposedContent *)content)->
                        getLabel()->size()];
      strcpy(label,((ComposedContent *)content)->
                  getLabel()->get());

   }
   else
   {
      label = new char;
      *label = '\0';
   }

   // create view
   node_view = bst_cbk->CreateNodeView(getBitmap(), label);

   // modifies display position if needed (visual restrictions)
   if (node_position.y < TOP_WALL_POSITION)
      display_position.y = TOP_WALL_POSITION;

   if (isPrivate())
   {
      if (node_position.x < PRIVATE_LEFT_WALL_POSITION)
         display_position.x = PRIVATE_LEFT_WALL_POSITION;
      if (node_position.x > PRIVATE_RIGTH_WALL_POSITION)
         display_position.x = PRIVATE_RIGTH_WALL_POSITION;
   }
   if (isPublic() && node_position.x <
                     PUBLIC_LEFT_WALL_POSITION)
     display_position.x = PUBLIC_LEFT_WALL_POSITION;

   // add view to the graph view

   bst_cbk->getGv()->
      addNode(display_position, node_view, node_id);

   if (is_interface_monitor_activated)
   {
      interface_change_monitor->setDisplayPosition
         (getInterfaceChangeMonitorViewPosition ());
      interface_change_monitor->openView();
   }

   if (is_concurrency_monitor_activated)
   {
      concurrency_monitor->setDisplayPosition
        (getConcurrencyMonitorViewPosition ());
      concurrency_monitor->openView();
   }

   view_state = VISIBLE;

}

/*-------------------------------------------------------------
 |
 | closeView
 |
 | Closes the view of the node. In this case, the edgar node
 | view is deleted, but the graph node is maintained.
 |
 *-----------------------------------------------------------*/
void EdgarNodeContainer::closeView()
{
   // invocation cannot be attended
   if(!hasView() || isClosed() || !bst_cbk)
      return;

   bst_cbk->getGv()->deleteNode(node_id);

   closeLabel();

   interface_change_monitor->closeView();
   concurrency_monitor->closeView();

   view_state = HIDDEN;
}
```

Figura 73 - Métodos particulares à aplicação para abertura e fecho da representação do Contentor

```
/*-------------------------------------------------------------
 |
 | class BrainStormingConnection
 |
 | Implements the BrainStorming application type checking for
 | the connection. Some connections are not available.
 |
 *-----------------------------------------------------------*/
class BrainStormingConnection : public Connection {

   public:

      BrainStormingConnection() : Connection() {}

      ~BrainStormingConnection() {}

      // Determines if the connection is allowed for
      // this application
      int connectionAllowed(Container *_destiny_hint = NULL);

      // calculates the destiny for free space destiny_hint
      Container *getDestiny();

      // gets the structural change type made by the connection
      int getStructuralChangeType();

      // tells if the origin is public or not
      int originIsPublic() { return origin->isPublic(); }

      // Connection phases
      int finish(Container *_destiny, Point position,
                 Bool is_a_remote_connection = FALSE);

      // Broadcast this interaction connection to all
      // participants
      void broadcastPublicToPublicInteraction
           (Container *destiny, Point position, Bool *gived_up);

      // Broadcast a lock container interaction connection, that
      // is, a move action from public to private space.
      void broadcastPublicToPrivateInteraction ();

      // Broadcast a container move/creation interaction
      // connection from private to public space.
      void broadcastPrivateToPublicInteraction ();

      // Sets the global and local node id relation
      void setTransferNodeRelationID (long global_node_id = 0);

      void broadcastStartConnection  ();
      void broadcastGiveUpConnection ();
      void waitForConflitInformation (Bool *is_in_conflit);

   protected:

      // structural change types
      enum { NO_CHANGE, NODE_CHANGE, NODE_CREATION };

      // Data to make visual relaxation
      Container *origin_referenced_by;
};
```

Figura 74 - Instanciação de Connection para a aplicação *IDEAGEN*