Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Um modelo de grupos para aplicações interactivas distribuídas

Carmen Pires Morgado

Dissertação apresentada para a obtenção do Grau de Doutor em Informática pela Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia.

Lisboa
(2007)

Esta dissertação foi desenvolvida sob orientação do
Professor José Alberto Cardoso e Cunha e
Professor Nuno Manuel Robalo Correia,
do Dept. de Informática da Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa.

- nº de arquivo
- "copyright"

Aos meus pais

Agradecimentos

Quero aqui agradecer a todos aqueles que de forma directa ou indirecta contribuíram para a realização do trabalho aqui apresentado.

Em particular, gostava de agradecer aos meus orientadores, Prof. Cardoso e Cunha e Prof. Nuno Correia por todo o apoio não só técnico como humano e pela disponibilidade demonstrada ao longo de todo o tempo de elaboração deste trabalho.

Quero também prestar um agradecimento muito especial aos amigos de sempre Miguel e ao João por toda a ajuda prestada ao longo da realização deste trabalho e principalmente pela paciência para me aturarem nos momentos mais complicados. Ao Luís Soares pelo ajuda no estudo dos suportes de desenvolvimento em particular na ajuda em "domar" a primeira plataforma de suporte utilizada (JINI). Ao Jorge Custódio, pela disponibilização da plataforma JGroupSpace e por todos os esclarecimentos prestados na utilização da mesma.

Um agradecimento também aos meus colegas de departamento em particular aos da secção de Arquitectura de Sistema Computacionais pelo apoio e suporte prestado durante o desenvolvimento deste trabalho. Quero agradecer também a todos os membros do grupo IMG pelo apoio e boa disposição que contribuiu para ajudar a superar as fases mais complicadas.

Quero ainda agradecer ao Departamento de Informática e ao CITI, as condições de trabalho oferecidas que tornaram possível a conclusão deste trabalho.

Finalmente um agradecimento muito especial para os meus pais, ao Manuel, Esmeralda e Teresa que sempre me apoiaram e sempre confiaram. Não posso também deixar de agradecer à Vimi pela boa disposição e pelos passeios que em muito contribuíram para manutenção do meu nível de *stress* em valores aceitáveis.

Resumo

A generalização da utilização de dispositivos móveis tais como portáteis, PDAs e telemóveis, com capacidades computacionais cada vez maiores, assim como a evolução que tem vindo a surgir na área das infraestruturas de rede sem fio ("wireless"), oferece aos utilizadores a possibilidade de estarem ligados entre si a qualquer momento. Este facto deu origem a um desenvolvimento significativo no surgimento de aplicações, que explorem as características de mobilidade e interacção. É também um facto que os seres humanos possuem uma tendência natural para interagirem entre si, estabelecendo comunicações como forma de partilhar informação e também estabelecer formas de relacionamentos estruturados, formando deste modo grupos de interesse. Isto motivou a investigação na área dos modelos, abstracções e mecanismos que permitam o estabelecimento de interacções de um modo transparente e flexível entre grupos de utilizadores. Neste trabalho são discutidos e analisados possíveis cenários de aplicação e os requisitos que estes colocam aos ambientes de suporte. Foi dada especial atenção à forma como as entidades nestes ambientes estabelecem interacções e se organizam em estruturas mais complexas (grupos), com o objectivo de partilhar, cooperar e colaborar entre si, com a definição de um novo conjunto de necessidades de serviços.

O principal objectivo deste trabalho foi o de criar e desenvolver um modelo capaz de captar a dinâmica das aplicações interactivas distribuídas, suportando as interacções entre os membros do grupo, de forma transparente e flexível. O modelo compreende um conjunto de primitivas especializadas para ambientes de apoio a aplicações baseadas em grupos. Estas primitivas ajudam o programador de aplicações a especificar os padrões de colaboração e comunicação entre os membros dos grupos. Os grupos podem ser criados e destruídos dinamicamente e a sua composição pode variar ao longo do tempo.

Summary

The generalization of the use of mobile devices such as laptops and hand-held devices with increased computational capabilities, as well as the evolution on the wireless network infrastructure have given the users the possibility of being connected all the time to each other. This gave place to a significant increase in the development of applications, exploring mobility and interaction. It's also a fact that humans tend to interact with each other, establishing communications in order to share information and naturally tend to establish relationships and form groups. This motivates research on the models, abstractions and mechanisms which enable a transparent and flexible specification of interactions between users and group based collaborations. In this work we discuss typical application scenarios and requirements they pose to the underlying support environments. We are particularly concerned with the way the entities, within these environments, establish connections and group themselves in more complex structures, defining new sets of functionalities and services.

The main goal of this work was to define and develop a model able to capture the dynamics of the application driven group management, in order to support the interactions between group members, in a transparent and flexible way. The model encompasses a set of primitives and services that are specialized for environments supporting group based applications. The primitives help the developer to specify the collaboration and communication between members of established groups. Usually these groups have a dynamic nature, with members entering and leaving, and groups being created and destroyed over time.

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

Conteúdo

Neste capítulo é feita a apresentação do tema e das motivações, dos objectivos e das principais contribuições, assim como da organização do texto da dissertação.

1.1 Enquadramento e motivação

O trabalho aqui apresentado centrou-se na concepção de um modelo e do seu ambiente de execução para o desenvolvimento de aplicações interactivas distribuídas. Estas aplicações possuem uma forte componente de interacção com o utilizador, podem suportar múltiplos utilizadores simultâneos cooperantes que partilham informação, sendo também de um modo geral aplicações que tendem a possuir um carácter móvel.

A mobilidade física foi tornada possível pelo surgimento e disseminação de um conjunto de novos dispositivos móveis (ex: telemóveis, PDAs, *notebooks*, etc.), assim como de infraestruturas de comunicação sem fios (baseadas em normas como o IEEE 802.11, Bluetooth), que facilitam e motivam o desenvolvimento de novos tipos de aplicações. Com a diversidade de dispositivos e de infraestruturas de comunicação que têm vindo a surgir, existe cada vez mais a necessidade de uniformizar a forma como se lida com esses elementos. As soluções que têm vindo a emergir apontam para a criação de arquitecturas de serviços que tenham a capacidade de simplificar a forma como as aplicações lidam com a heterogeneidade dos dispositivos e das infraestruturas de comunicação (e mesmo dos perfis dos utilizadores). No entanto, dadas as variações no comportamento, nas características dos dispositivos e na sua inserção em ambiente móvel, torna-se difícil assegurar, de uma forma rígida e definitiva, a funcionalidade desejada em cada caso específico, mantendo ainda uma qualidade de serviço aceitável para cada cenário particular de aplicação. Isto originou investigação e desenvolvimento de plataformas de suporte flexíveis, que permitam ao programador desenvolver aplicações capazes de reagirem dinamicamente e adaptarem-se a alterações no comportamento dos componentes do ambiente.

Uma das principais características das aplicações interactivas móveis é a sua natureza dinâmica, em termos dos utilizadores e das entidades computacionais envolvidas e das suas interacções. Em particular, estas interacções podem ser estabelecidas de forma espontânea e sem um plano previamente estabelecido, podendo levar à constituição de comunidades colaborativas, que assentam na partilha de informação.

Como forma de modelar a cooperação e a coordenação entre as entidades envolvidas, esta tese propõe uma abordagem baseada num modelo de grupos. A problemática dos grupos tem sido objecto de investigação intensa, nas últimas décadas [Bir05]. Nela se inserem o estudo de dimensões tão diversas como a noção ou consciência de grupo, os modelos de cooperação e organização dinâmica de aplicações distribuídas e a tolerância a falhas. Em particular, tem havido desenvolvimentos na aplicação de modelos de grupos para suportar ambientes colaborativos distribuídos e têm existido diversos esforços na sua adaptação a ambientes móveis.

Um dos objectivos centrais desta tese é a integração de certas funcionalidades

características dos modelos de computação distribuída baseados em grupos de processos, no contexto de aplicações interactivas em ambientes de computação móvel. Exploram-se as abstracções de grupos, como forma de modelar, não só a organização dinâmica das entidades computacionais, como as suas interacções.

Este trabalho enquadra-se em direcções de investigação em curso no CITI - Centro de Informática e Tecnologias de Informação, Departamento de Informática, FCT/UNL, nas linhas de Processamento Paralelo e Distribuído e de Processamento e Visualização de Informação Multimédia. Este trabalho prossegue investigação anteriormente desenvolvida em modelos para a estruturação de aplicações distribuídas, baseados em grupos, e em sistemas de computação multimédia móvel.

1.2 Objectivos

O objectivo principal desta dissertação é a definição e desenvolvimento de um modelo capaz de captar a dinâmica que envolve o estabelecimento de grupos, de modo a suportar as interacções entre os membros dos grupos de um modo transparente e flexível. O modelo engloba um conjunto de primitivas e funcionalidades especializadas para suportar as necessidades das aplicações que lidam com grupos de utilizadores que interagem através de entidades computacionais. As primitivas auxiliam o programador de aplicações na especificação dos padrões de comunicação e de colaboração que se estabelecem num ambiente de grupos e em sistemas de computação distribuída. Os grupos de utilizadores, como é usual nos grupos sociais possuem uma natureza dinâmica, com membros a entrarem e a sairem e os próprios grupos a serem criados e cancelados ao longo do tempo.

Ao longo dos estudos que foram desenvolvidos no âmbito deste trabalho, observou-se que de um modo geral os seres humanos tendem a utilizar os grupos de duas formas distintas:

- através de demonstração explícita da sua vontade de passar a pertencer ou abandonar o grupo (formar ou destruir grupos);
- ou através da criação implícita de comunidades constituídas com base nas preferências dos utilizadores e em regras definidas e características de cada aplicação.

O modelo proposto suporta a definição destas duas classes de grupos, que foram denominados de "grupos explícitos" e de "grupos implícitos", tendo estes funções e utilizações distintas. Cada grupo pode ter associado um conjunto de conteúdos produzidos pelos membros e que ficam acessíveis a todo o grupo.

No caso dos grupos explícitos existem diversas políticas de admissão (filiação) de membros, sendo este facto reflectido na proposta do modelo, onde na configuração (quando da criação) fica definido o tipo de filiação a que os membros ficam sujeitos.

Os grupos implícitos são grupos formados "automaticamente" com base num conjunto de características e/ou preferências que deve ser observado por todos os membros. Estes grupos podem ser formados, por exemplo, com base na posição ocupada pelos membros no espaço, ou podem ser utilizados como forma de criar comunidades que partilhem o mesmo tipo de interesses e que se encontram na proximidade. Os grupos implícitos pode ser utilizados, por exemplo, num espaço comercial onde o administrador do espaço pode optar por definir os grupos de utilizadores que possuem interesses comuns. Um exemplo de grupo implícito definido nesse espaço pode ser o grupo de utilizadores que gostam e praticam algum tipo de desporto. Este grupo pode depois ser utilizado como forma de divulgação de informação referente, por exemplo, a promoções de artigos desportivos.

1.3 Dimensões do problema

São referidas nesta secção as dimensões que foram envolvidas no desenvolvimento deste trabalho, tais como:

- as aplicações;
- os modelos e abstracções de grupo;
- as plataformas de serviços e infraestruturas.

1.3.1 Aplicações

As aplicações consideradas neste trabalho colocam em jogo entidades computacionais distribuídas ou seja, utilizadores que acedem a dispositivos físicos de interacção e cujas interacções são representadas e geridas por componentes de software. As seguintes características principais devem ser tidas em conta:

- diversidade e heterogeneidade, incluindo a multimodalidade das interfaces e os diferentes tipos de interacções que se estabelecem entre as entidades;
- localização;
- mobilidade;
- escala;
- qualidade de serviço.

Ao nível das aplicações, as interacções entre as entidades envolvidas são dinâmicas e existe uma grande diversidade e heretogeneidade de tipos de dispositivos e perfis de utilizadores. Na semântica destas interacções existe uma grande dependência do

contexto envolvente da aplicação. Esse contexto pode ser definido ao nível do tipo de dispositivo, do tipo de utilizador e das permissões que lhe estão atribuídas, ou mesmo das características do meio envolvente, como a localização, hora, altura do ano ou o estado do tempo. Outra vertente das características das aplicações consideradas é a forma como se estabelecem dinamicamente as interacções entre utilizadores e como estes se organizam de forma mais ou menos estruturada. Neste tipo de aplicações, a forma de estabelecimento da interacção entre utilizadores pode variar e ser definida de forma pré-determinada e pró-activa, de forma mais ou menos oportunística ou espontânea, associada a oportunidades que surgem quando os utilizadores descobrem afinidades (de proximidade, de interesses ou objectivos) e disponibilidade mútua para a interacção.

Neste trabalho, também se contempla uma dimensão que é típica das aplicações sensíveis ao contexto ("context awareness"). Este tipo de aplicações explora a mobilidade dos utilizadores para, em função dos seus perfis de interesses, procurar disponibilizar a informação adequada ao contexto corrente. É o caso, por exemplo, de um turista que pretende descobrir um determinado ponto de interesse no seu itinerário, com recurso a um guia turístico interactivo. De igual modo, ao envolverem comunidades dinâmicas de utilizadores, estas aplicações procuram explorar as oportunidades de estabelecimento de interacções mútuas, proporcionando o suporte desejado para a partilha de informação.

1.3.2 Modelos e abstracções de grupo

Tem havido intensa investigação e desenvolvimento em aplicações suportando trabalho colaborativo [YWLC99, KMD02], dirigidas a grupos de utilizadores distribuídos [RCH+99, Per02, MSW03], com gestão da consistência da informação partilhada e permitindo interacções síncronas ou assíncronas, consoante a semântica das aplicações.

O surgimento de dispositivos computacionais móveis e de suporte à comunicação sem fios permitiu explorar as potencialidades dos sistemas de *groupware* em novos contextos de aplicação. As soluções tradicionais dos sistemas baseados em suportes de comunicação fixos e mais confiáveis não são directamente aplicáveis aos sistemas móveis, dadas as limitações intrínsecas dos dispositivos e as características das comunicações, afectando a disponibilidade individual dos dispositivos e a sua conectividade. Isto tem motivado diversos esforços para fazer evoluir o suporte à colaboração e interacção entre entidades móveis, no sentido de dar um maior conhecimento ou consciência (*awareness*) e um maior controlo, à própria aplicação,

relativamente às variações no comportamento dos diversos parâmetros que afectam a qualidade de serviço que pode ser disponibilizado a cada momento. Tal evolução exige uma análise, dependente de cada tipo de aplicação, das suas características relativamente à complexidade e tipo de interacções, ao grau de consistência exigido, às restrições de tempo (*deadlines*) a satisfazer, ao número e localização dos participantes envolvidos, entre outras.

Esta abordagem tem implicações a nível das funcionalidades suportadas pelas plataformas de serviços de grupos, por exemplo, relativamente à transparência na gestão dos espaços partilhados e às semânticas de consistência dos estados globais observados pelos membros de um grupo. Os desenvolvimentos que se têm observado vão no sentido de uma maior flexibilidade, para permitir a adaptação da própria aplicação, face às modificações nos tipos das interacções, dependentes do momento temporal, da localização dos intervenientes e dos seus contextos. As soluções a adoptar têm também de permitir adaptação ao nível das modificações nos ambientes de execução, dada a diversidade, heterogeneidade e dinamismo dos dispositivos envolvidos.

Nesta tese explorou-se uma direcção de investigação que tem vindo a ser desenvolvida neste departamento, relativamente a abstracções de grupos, para a estruturação e organização dinâmicas de aplicações distribuídas. O trabalho previamente desenvolvido situava-se a um nível de abstracção mais elevado e está actualmente concretizado no modelo GroupLog [Bar04], que dispõe de uma especificação numa linguagem lógica. Esse modelo suporta o estabelecimento dinâmico de grupos, possivelmente hierárquicos, como unidades de cooperação entre entidades distribuídas, cujos membros podem interagir directamente entre si ou através de espaços logicamente partilhados, internos ao grupo.

Um dos objectivos centrais desta tese é o estudo das diversas problemáticas da adaptação e da evolução daquele modelo de grupos para o desenvolvimento de aplicações do tipo das acima descritas, visando melhorar as funcionalidades das plataformas de suporte, necessárias para o desenvolvimento dessas aplicações.

1.3.3 Arquitecturas de serviços e infraestruturas

Têm-se verificado diversos desenvolvimentos nas arquitecturas de serviços, no sentido de permitir a coexistência de diferentes dispositivos, que podem partilhar e aceder a informação e serviços de uma forma simples e transparente, possibilitando a partilha de recursos, disponibilizados pelas diversas entidades presentes num dado momento.

Os protocolos de descoberta de serviços têm como objectivo apoiar a reconfigu-

ração dos recursos existentes ao nível da arquitectura e da infraestrutura de rede, lançando e publicando serviços dinamicamente, de modo a melhorar e simplificar a utilização por parte dos utilizadores móveis.

Outros aspectos que são contemplados a este nível são a uniformização das interfaces, a interoperabilidade e a independência da plataforma face às infraestruturas e os aspectos de segurança, que deverão suportar métodos para autenticação e autorização de todas as entidades envolvidas e que forneçam segurança no transporte de dados.

O suporte de aspectos como o apoio à cooperação, organização dinâmica, adaptabilidade, mobilidade, escala (em termos do número de entidades envolvidas) foram analisados, em diversas plataformas de serviços existentes, no sentido de se avaliarem as soluções mais adequadas para os desenvolvimentos propostos neste trabalho.

Relativamente à infraestrutura de suporte, situam-se as funcionalidades fornecidas pelas redes móveis *ad-hoc* (também denominadas de espontâneas), cuja topologia pode sofrer modificações, rapidamente e de formas não previstas. Este tipo de redes pode permanecer ou existir isolada ou possuir ligações a outro tipo de rede. Tem como vantagem a característica de ser automaticamente configurável e de se poder adaptar facilmente às modificações, quer dos seus elementos, quer de outras características, como por exemplo, condições de transmissão, tráfego e equilíbrio de carga[1].

Estas características levantam no entanto diversos problemas que estão essencialmente relacionados com o facto de a topologia da rede ser instável, devido à movimentação dos seus nós (elementos). Várias abordagens têm sido propostas para resolver os diversos problemas deste tipo de rede/configuração.

1.4 Metodologia

Este trabalho contemplou três aspectos principais, relacionados entre si, que orientaram os desenvolvimentos que conduziram à escrita da dissertação:

1. investigação das abstracções que facilitam a estruturação/organização de sistemas em grupos suportando a interacção e cooperação entre entidades dinâmicas, em aplicações interactivas;

2. concepção e realização de soluções que permitiram concretizar as abstracções sobre plataforma/infraestrutura existentes, num ambiente distribuído;

3. validação do modelo e da sua realização, face a um conjunto de aplicações seleccionadas.

[1]Este tipo de infraestrutura possibilita o desenvolvimento de aplicações vulgarmente denominadas de "anytime, anywhere applications".

O primeiro aspecto prosseguiu investigação anterior, que conduzira ao modelo GroupLog, e teve como principal objectivo o estudo da sua adaptação ao domínio da aplicações mencionadas. Desse estudo resultou uma proposta de um modelo (MAGO), que contempla os requisitos identificados.

O segundo aspecto investigou as possíveis realizações do modelo, sob a forma de extensões ou camadas suportadas por plataformas de serviços existentes. Dessa investigação resultou a concepção e implementação de um protótipo experimental, que permitiu validar as propostas efectuadas.

O terceiro aspecto contemplou a validação do modelo, face a aplicações concretas.

As fases principais do plano de trabalho seguido ao longo desta dissertação, foram:

1. caracterização do estado da arte do domínio das aplicações interactivas distribuídas, incluindo os modelos das aplicações, as plataformas de serviços e as infraestruturas de comunicação;

2. investigação das dimensões a contemplar num modelo de grupos, para permitir formas de organização dinâmica de entidades computacionais, com interacção directa, por mensagens e por estado partilhado;

3. integração dos mecanismos suportando as abstracções do modelo acima investigado, numa arquitectura desenvolvida sobre uma plataforma de programação distribuída existente;

4. desenvolvimento de um protótipo que permitiu validar o modelo/arquitectura proposta, e aplicação do modelo/arquitectura em aplicações já existentes;

5. escrita da dissertação.

1.5 Contribuições

Esta investigação tem como principal contribuição a definição de um modelo de grupos que permita a estruturação de aplicações interactivas distribuídas e a sua realização através de uma arquitectura, suportada por um protótipo experimental. Este modelo contempla os seguintes aspectos:

- definição e respectiva caracterização de entidades elementares e de entidades compostas (grupos);

- definição de mecanismos que permitam captar os padrões de estruturação e de interacção entre entidades;

- utilização dos conceitos definidos para a modelação e desenvolvimento de aplicações com necessidades de comunicação e partilha de dados entre utilizadores.

Foi concretizada uma implementação do modelo proposto, que foi denominado de MAGO - *"Modeling Applications with a Group Oriented approach"* [MCC07]. Esta realização possibilitou o ensaio experimental da aplicação e proceder também à validação da proposta de modelo apresentada. Foi também avaliada a utilização desta proposta face a aplicações já existentes e a forma como os novos padrões de organização e interacção permitem a definição de novas funcionalidades nessas aplicações.

1.6 Organização

Neste primeiro capítulo são apresentados os objectivos, motivações e principais contribuições desta dissertação, bem como o seu enquadramento em termos de áreas de investigação envolvidas.

Nos capítulos 2 e 3 é feito um levantamento das áreas afins, que estão mais directamente relacionadas com os temas abordados ao longo desta dissertação. Em particular, no capítulo 2, são descritas e analisadas aplicações interactivas existentes, identificando-se as suas características e necessidades particulares, assim como dos diversos domínios possíveis de aplicação. No capítulo 3, são descritos modelos de comunicação básicos que possuem características e propriedades que permitem dar suporte à implementação dos ambientes das aplicações referidas.

No capítulo 4, é apresentada a proposta do modelo, denominado de MAGO - "Modeling Applications with a Group Oriented approach". Com este modelo, pretende-se capturar o conjunto de propriedades e funcionalidades das entidades intervenientes num sistema de grupos, pretendendo-se, com este modelo, captar os padrões de interacções e comunicações típicos nas estruturas de grupos. No final deste capítulo são descritos cenários de possíveis utilizações do modelo proposto, em ambientes com necessidades reais de interacção entre utilizadores, bem como os benefícios da sua estruturação em grupos.

No capítulo 5, é descrita a realização do modelo e a especificação da arquitectura que o suporta.

No capítulo 6, é descrito um protótipo de um aplicação que foi estruturada e desenvolvida com base na arquitectura proposta e que permitiu validar as primitivas definidas e ainda avaliar as vantagens de interesse prático das funcionalidades e mecanismos disponíveis. Neste capítulo é também analisada a aplicabilidade do modelo proposto no desenvolvimento de novas funcionalidades em aplicações já existentes.

Finalmente, no capítulo 7, são apresentadas as conclusões e são identificadas áreas para futuros desenvolvimentos, em particular algumas possíveis utilizações da abordagem, para melhorar as funcionalidades em aplicações já existentes em diferentes domínios.

2

Interacções entre múltiplos utilizadores

Conteúdo

Nesta secção é feito um levantamento relativo à evolução dos sistemas de computação e comunicação e à sua utilização em aplicações que promovem a interacção. São definidos conceitos e descritas aplicações que ilustram os novos mecanismos de interacção entre grupos de utilizadores e a forma como estes têm influenciado o modo como comunicamos, estabelecemos e mantemos contacto dentro de grupos. São também caracterizados e identificados algumas das necessidades e requisitos particulares das aplicações que lidam com a interacção entre múltiplos utilizadores.

2.1 Introdução

Os seres humanos interagem entre si e comunicam de forma a partilhar informação, estabelecendo relações e formando grupos como modo de atingir um determinado objectivo comum. Estes objectivos podem ser os mais diversos, quer de origem profissional, social ou pessoal [Que05] e estão associados, de um modo geral, com necessidades de colaboração e partilha de informação entre múltiplos participantes.

A divulgação das tecnologias, em particular associadas à Internet e aos dispositivos móveis levou ao surgimento de novas plataformas e aplicações que motivam e facilitam os processos de interacção não presencial entre indivíduos.

Ao longo deste capítulo é feito um levantamento de alguns dos conceitos relacionados com as novas formas de interacção, em particular no que se refere à forma como estas promovem o estabelecimento de estruturas e de redes sociais. Paralelamente são referidos alguns exemplos de aplicações que fomentam a interacção e partilha de dados entre os utilizadores e a forma como estas podem ser utilizadas em diversos domínios. Finalmente, são enumerados os principais desafios e questões que se colocam a estas aplicações e à sua utilização em diferentes ambientes, com especial ênfase nos problemas relacionados com a interacção entre utilizadores e a sua agregação em estruturas sociais, bem como a sua utilização em ambientes distribuídos móveis.

Como já foi referido, o desenvolvimento que se tem vindo a observar ao nível das infra-estruturas de comunicação e dos dispositivos e a vulgarização da sua utilização, tem levado a um aparecimento crescente de novas aplicações com o objectivo de promover a interacção entre utilizadores. Tem-se vindo a observar uma especialização e adaptação dos serviços disponibilizados pelas aplicações existentes, sendo utilizadas num número cada vez maior de domínios e por um número crescente de pessoas.

Ao longo das secções seguintes são identificados e caracterizados três domínios de aplicações que têm vindo a surgir na área das interacções entre utilizadores: (1) interacção em redes sociais, (2) partilha e troca de informação e (3) interacção em ambientes móveis. De notar que, embora seja efectuada esta diferenciação, muitas das aplicações que vão ser referidas como exemplo integram funcionalidades dos três tipos mencionados.

2.2 Interacção em redes sociais

Ao nível da interacção em rede tem-se observado um enorme desenvolvimento e optimização de serviços e aplicações na área do denominado "software social"[1], com uma generalização crescente da sua utilização nos mais diversos tipos de situações. Este tipo de aplicações tem alterado a forma como os seres humanos estabelecem os seus

[1]Tradução directa do inglês "*social software*".

contactos e se relacionam entre si, criando novas formas de interacção e comunicação.

O termo "software social" é normalmente utilizado para designar o conjunto de programas (suportados na Web de um modo geral), que permitem a interacção entre utilizadores para partilharem informação e comunicarem entre si. Neste tipo de interacção, as comunicações são mediadas por computadores e podem recorrer a diversos tipos de *media* (texto, áudio, vídeo) [wik07c, wik07b]. Podem ser consideradas, dentro desta classe, aplicações que vão desde o *email* ou *chats* até aplicações de encontros ou software de suporte à colaboração. A aplicação do termo "software social" está de certo modo mais ligada à forma como são utilizadas as aplicações, do que com algum tipo de características particulares das próprias aplicações. No entanto, de um modo geral, as aplicações que se situam dentro deste tipo de classificação promovem e facilitam a interacção entre utilizadores de um modo não presencial.

Pode-se dizer que o termo "software social" tem vindo a ser utilizado para designar, não só um conjunto de produtos, mas também a forma como os utilizadores interagem com recurso a estes. Segundo uma definição de Clay-Shirky[2], trata-se de: "Software que suporta a interacção em grupo" [Shi07]. Esta definição aproxima-se da definição de *groupware*[3] [YWLC99], área na qual se que têm vindo a desenvolver aplicações, plataformas e ferramentas de apoio à colaboração e cooperação entre utilizadores.

O denominado Software Social, em particular os *sites*[4] que disponibilizam serviços para facilitar a interacção e comunicação entre utilizadores, representam uma nova geração de tecnologias para a descoberta e o estabelecimento de contactos entre utilizadores.

As aplicações existentes para suportar este tipo de interacções não presenciais possuem, de um modo geral, funcionalidades que permitem aos utilizadores definirem o seu perfil, assim como gerir a sua rede de contactos e suportar comunicação de grupo. São serviços centrados, de um modo geral, em interacções através da Web e recorrem na grande maioria dos casos a comunicações assíncronas, em que as interacções entre os diversos participantes são efectuadas através de mecanismos de publicação de mensagens ou informações, que podem ser mais tarde consultadas.

As interfaces e os serviços existentes neste tipo de aplicações (em particular nos *sites* de serviços), são realizados e organizados de modo a facilitar aos utilizadores a consulta da sua rede de contactos e a colocação da sua informação. Possuem, de um modo geral, interfaces através das quais é possível, ao utilizador, configurar o perfil de acordo com as suas preferências, podendo essa acção ser efectuada quando do registo ou posteriormente.

[2]Clay-Shirky foi co-organizador do encontro "Social Software Summit" em 2002 onde apresentou os seus trabalhos, na área das interacções sociais.

[3]De um modo geral o termo *groupware* refere-se a software que auxilia a colaboração entre diversas pessoas que se encontram em diferentes locais, oferecendo mecanismos de coordenação e registo de alterações efectuadas.

[4]São geralmente denominados de SNS - "Social Network Sites" na literatura anglo-saxónica.

2.2.1 Sistemas e aplicações

Tem surgido, nos últimos anos, um grande número de *sites* que disponibilizam serviços para o estabelecimento de redes de contactos [BE07]. Como exemplos mais representativos de alguns destes *sites* de serviços podem ser referidos o MySpace, Hi5 e o FaceBook que possuem, actualmente, milhões de utilizadores em todo o mundo. A particularidade especial do conjunto de serviços existentes nestes *sites*, é o facto de possibilitarem aos utilizadores um acesso rápido e simples aos seus contactos directos, permitindo estabelecer e manter as redes sociais em que estes se encontram integrados.

Embora actualmente exista uma grande variedade de SNS - "Social Network Sites", de um modo geral, estes disponibilizam o mesmo tipo de funcionalidades e serviços aos seus utilizadores incluindo a definição do perfil, que fica acessível através de uma página Web, consulta da lista de "amigos"[5] [BE07], convite a novos membros e troca de informação. Para se filiarem num SNS, os utilizadores têm que se registar, consistindo este registo no preenchimento de um formulário[6]. A partir da informação recolhida, é gerada uma página com o perfil, que fica visível e com o qual um membro se apresenta aos restantes. O modo como é desencadeado o processo de filiação pode ter origem numa acção explícita por parte do utilizador ou pode surgir como resposta a um convite explícito desencadeado por um membro já registado, sendo mais uma vez este processo dependente do tipo de aplicação e utilização pretendida. Também a forma e o conteúdo do perfil que é tornado disponível por cada utilizador pode ser configurado de acordo com a filosofia de utilização da aplicação. Por exemplo, no caso do LinkedIn [web07c], a informação e os serviços disponíveis variam consoante se trata de um conta paga ou não. Em muitos casos, a informação e a forma como esta é visualizada depende da configuração e permissões de acesso definidas pelo próprio utilizador, podendo ter como opção o controlo da visibilidade da informação, para todos ou somente para o seu grupo de contactos autorizados (amigos).

Os primeiros SNS que surgiram tinham como objectivo estabelecer uma rede de contactos com um caracter genérico, tendo um conjunto de serviços básico de funcionalidades que englobavam a criação de perfis e a visualização da lista de contactos autorizados, sendo ainda possível a troca de mensagens entre utilizadores. O primeiro SNS com as características de serviços apresentadas foi o sistema SixDegrees (1997-2000) [BE07].

Paralelamente ao surgimento dos primeiros SNS, existiam já diversas aplicações que promoviam o estabelecimento de interacções entre utilizadores remotos como por exemplo o ICQ ou o mIRC,[7] que permitiam a troca directa de mensagens em tempo real

[5] O termo "amigos" aqui possui o significado de conjunto de utilizadores registados que pertencem ao seu grupo de contactos.

[6] O conjunto de campos obrigatórios, depende da aplicação em causa, embora de um modo geral seja pedida informação genérica (como por exemplo *email*, nome, idade, sexo, profissão, morada).

[7] Inicialmente tratavam-se de simples aplicações com interfaces de clientes baseadas no protocolo de comunicação IRC-Internet Relay Chat, para mensagens e arquivos entre utilizadores.

entre utilizadores registados. Mais recentemente, dentro desta linha de aplicações para troca directa de informação entre utilizadores surgiram os sistemas MSN da Microsoft, Skype[8], Google Talk[9] e o Yahoo Messenger.

Tem-se ainda assistido a um surgimento de SNS mais especializados, ao contrário dos primeiros que eram direccionados para um público mais heterogéneo, onde era atribuída aos utilizadores a responsabilidade da formação de grupos mais especializados. A própria evolução da utilização das redes leva à especialização de SNS que inicialmente surgiram com um carácter mais generalista. Um exemplo disso é o MySpace, que se direccionou mais para a área das bandas musicais e artistas de um modo geral, incluindo serviços especializados para este tipo de utilizadores e para os seus fãs [Boy07].

Nos últimos anos têm no entanto vindo a surgir SNS com uma natureza mais especializada logo de raiz, o que permite à partida disponibilizar um conjunto de serviços mais adaptados a uma determinada temática e tipo de utilizadores. Como exemplo desse facto, temos o LinkedIn (2003) direccionado para estabelecimento e divulgação de redes de contactos profissionais; o Dogster e o Catster (2004) direccionado para os donos e seus animais de estimação (cães e gatos respectivamente), sendo possível criar uma página Web através da qual é disponibilizada informação (sob a forma de texto, imagens e vídeos) referente aos animais; e o Geni (2007) que tem por base a estrutura familiar e permite a criação de árvores genealógicas.

Ainda dentro desta linha de aplicações (redes sociais), podem ser referidas as aplicações com uma forte componente gráfica, como é o caso por exemplo do Second Life (*www.secondlife.com*). Nesta aplicação, os utilizadores são representados num mundo virtual de forma simbólica por avatares, sendo através destes que interagem e estabelecem comunicação com os restantes elementos[10] desse mundo. Os utilizadores podem estabelecer contacto com outros participantes através de uma abordagem directa aos seus representantes, através de diálogo (som) ou mensagens de texto.

Outro exemplo de aplicação com uma forte componente gráfica, é o jogo de fantasia medieval "Dungeons and Dragons" (*http://www.ddo.com/*), em que os jogadores criam os seus próprios personagens através dos quais participam em aventuras, combatendo inimigos, procurando tesouros e interagindo também com outros participantes, conseguindo assim ganhar pontos que os tornam mais "fortes", para os futuros combates. Este constitui um exemplo dos denominados RPG (Role-playing Games), que são jogos onde os jogadores assumem os papéis de personagens e criam histórias

[8]Skype é um software que permite comunicação pela Internet através de conexões sobre VoIP (Voz Sobre IP)

[9]Google Talk é um serviço de mensagens instantâneas e de VoIP desenvolvido pela Google, baseado no protocolo aberto Jabber (também conhecido por XMPP), cuja primeira versão foi lançada em Agosto de 2005.

[10]Os elementos aqui referidos podem ser representações de outros utilizadores ou objectos existentes no meio.

e narrativas de forma colaborativa, estabelecendo ao longo da evolução do jogo, diferentes relacionamentos e interacções com os restantes participantes. A evolução do jogo processa-se de acordo com um sistema de regras pré-definidas a partir do qual os jogadores podem ir improvisando as acções, condicionando desta forma o modo como o jogo se desenrola [wik07a].

2.2.2 Características e funcionalidades típicas

As múltiplas aplicações referidas evidenciam um conjunto de características típicas, em particular as que envolvem interacções no contexto de grupos de indivíduos e que são relacionadas com a forma como estabelecem e gerem as suas interacções. Algumas das características e funcionalidades identificadas são descritas de seguida:

- Registar os participantes
  - novos membros, é pedida informação ao utilizador de modo a validar a sua entrada e é-lhe atribuída uma identificação única, de modo a garantir que cada elemento é único no sistema;
  - membros já registados, devem fornecer a informação (identificação) de forma a validar o seu acesso;
- Identificar contactos
  - através de identificação e informação própria dos indivíduos pertencentes ao sistema, sendo esta a situação em que se pretende encontrar um determinado participante;
  - através de uma ou mais preferências/características dos participantes. Nesta situação, de um modo geral, é retornado o conjunto de indivíduos que verificam essas preferências/características ou pode ser retornado um grupo, sendo este visto como uma entidade do sistema que agrega os vários indivíduos que possuem essas propriedades;
  - através da rede de contactos própria dos participantes, por exemplo consultar os "amigos" dos "amigos";
- Definir/estabelecer formas de interacção
  - Topologia da interacção
    - de 1 para 1, comunicação directa entre dois participantes;
    - de 1 para n, comunicação, de um modo geral, anónima quando se pretende por exemplo divulgar informação ou seja efectuar um anúncio para toda a comunidade;

    - comunicação no seio de grupo, esta é uma forma de comunicação mais restrita que a anterior dado que se pretende transmitir informação para um subgrupo dos utilizadores;
  - Coordenação temporal
    - síncrona (*online*), este tipo de comunicação acontece quando existe uma interacção em tempo real entre os vários interlocutores. Essa interacção pode ser efectuada através da troca de mensagens (tipo *chat*), através de audio ou vídeo (como por exemplo nas vídeo-conferências);
    - assíncrona (*offline*), neste tipo de comunicação existe uma "distância" temporal entre os interlocutores envolvidos, sendo de um modo geral enviada uma mensagem para o(s) destinatário(s) os quais podem recebê-la, como por exemplo no caso do *email*, através de uma acção directa de leitura;
  - Tipo de *media*, a informação trocada entre os interlocutores da comunicação pode incluir diferentes tipos de *media*, como por exemplo texto, audio, imagens, vídeo ou composições destas formas.

- Decidir/aplicar políticas de gestão das interacções
  - admissão/permissão de entrada e saída de membros dos grupos em que participam. A entrada de participantes pode ser sujeita a diferentes políticas de admissão, por exemplo:
    - por convite explícito por parte de algum participante já membro;
    - por entrada directa desencadeada por acção do próprio interessado (bastando, de um modo geral, que seja um participante registado no sistema);
    - por possuir o conjunto de características que o validam como membro;
    - por uma candidatura a membro, que pode ser sujeita a uma validação por parte de um ou mais membros;
  - gestão das preferências e do perfil. Para além do conjunto de dados próprios, que são de um modo geral pedidos, é pedido aos membros que definam um conjunto de preferências que os caracterizem, sendo esta informação utilizada como forma de personalizar a aplicação e a informação disponibilizada ao utilizador e também como forma de sugerir possíveis grupos e interesse;
  - duração/persistência do meio de suporte à interacção, pode existir devido à manutenção local a cada utilizador de um histórico de comunicações efectuadas por este, ou pode ser mantido o estado das comunicações pelo sistema. No entanto, de uma forma geral as conversações de grupos de utilizadores,

só podem ser "recuperadas" no caso do utilizador se encontrar activo (*on-line*) no momento das mesmas. Em diversos sistemas observou-se que os próprios grupos de comunicação só tinham existência durante o tempo em que os interlocutores se encontravam activos no sistema, sendo o grupo encerrado após a saída de todos os participantes, ou seja tratam-se de grupos voláteis.

2.3 Partilha de informação

Embora este tópico seja apresentado separadamente, este tipo de funcionalidades constitui um dos serviços normalmente também disponibilizados pela classe de aplicações anteriormente referidas. De facto, de um modo geral, as aplicações de redes sociais disponibilizam serviços para a partilha de dados entre utilizadores e, do mesmo modo, as aplicações que suportam a partilha de informação também permitem o estabelecimento de interacções. As principais diferenças situam-se na forma como os serviços e as funcionalidades são apresentados ao utilizador. Aplicações como Hi5 e Facebook, por exemplo, possuem funcionalidades que permitem aos utilizadores partilharem e anotarem fotos. No entanto, o seu foco principal não é esse mas sim o de facilitar e promover a interacção entre membros e a formação de grupos de membros com afinidades entre si.

O objectivo principal das aplicações de partilha e troca de informação é o de proporcionar aos utilizadores uma forma de organizarem os seus conteúdos, tais como fotos ou vídeos e de os tornar acessíveis. Este tipo de aplicações, com um carácter genérico, permitem a colocação de conteúdos (com meta-informação associada) por parte de qualquer utilizador, podendo depois serem acedidos com base em palavras chave que foram associadas aos conteúdos.

2.3.1 Sistemas e aplicações

Ao nível da partilha de fotos existem diversas aplicações Web, que permitem aos utilizadores colocarem as suas fotos e torná-las disponíveis a outros utilizadores. Mais do que funcionarem somente como um repositório de fotos, aplicações como Flicker, Ringo ou Zurfer da Yahoo, permitem organizar e catalogar as fotos de acordo com as preferências dos utilizadores, definir diversos grupos de permissões de acesso e também colocar comentários sobre as fotos existentes, assim como deixar mensagens que podem ser lidas pelos utilizadores. As permissões de acesso à informação, assim como o acesso aos comentários são definidas, de um modo geral, pelo produtor da informação ou pelo conjunto de utilizadores a que este deu permissões para consulta. Para além das funcionalidades directamente associadas à partilha e organização de

fotos, estas aplicações Web possibilitam a interacção entre grupos de utilizadores, não só através da informação partilhada mas também através de possibilidade de comunicação directa entre utilizadores.

Em termos de partilha de conteúdos de vídeo, a aplicação Web mais utilizada é o YouTube, onde os utilizadores registados podem colocar vídeos que podem ser consultados por quem aceder ao *site*. Os utilizadores, para além de visualizarem os vídeos existentes, podem também deixar comentários e votar nos vídeos de que mais gostarem. Esta aplicação tem associado um mecanismo de pesquisa que permite a consulta baseada em temas e palavras chave, que foram definidos quando da colocação do vídeo por parte do produtor. Para colocar (*upload*) vídeos, o utilizador deve estar registado, tendo acesso a um conjunto de funcionalidades que lhe permitem configurar as permissões de acesso aos comentários do vídeo.

As aplicações para a partilha de fotos e vídeos mencionadas têm um caracter genérico. Tal como foi referido, existe no entanto um conjunto de aplicações com um carácter mais especializado, que disponibilizam conjuntos de funcionalidades directamente relacionadas com um tema e/ou domínio especifico. Um exemplo interessante de utilização deste tipo de funcionalidades é o de um sistema de informação e auxílio à navegação Web de pessoas com deficiências de locomoção denominado de "canal*ACCESSIBLE" [web07a], que foi desenvolvido para a cidade de Barcelona. Este sistema gere a informação captada por diversos utilizadores através de telemóveis, sendo essa informação composta por foto e local (morada) onde se verifica que existe um problema de acessibilidade. Essa informação é depois colocada num mapa da cidade que pode ser consultado através da Web. Os locais onde foram captadas as fotos podem ser consultados através de uma função de pesquisa, que permite definir um "novo" mapa da cidade, com sinalização dos locais onde a circulação de pessoas com deficiências de locomoção se torna difícil ou mesmo impossível. Este exemplo revela-se especialmente interessante devido ao seu domínio de aplicação e tipo de utilizadores.

Outro exemplo interessante de aplicação é o CONFECTIONARY desenvolvido pelo MIT MediaLab [Med07]. Esta aplicação permite criar "narrativas" com base em conteúdos (imagens e vídeos) e textos associados de forma dinâmica. Pode, por exemplo, ser utilizado como forma de criar um diário de viagem personalizado.

Diversas aplicações anteriormente referidas na secção 2.2 permitem a partilha de conteúdos multimédia, como por exemplo FaceBook ou mesmo Dogster, onde pode ser colocada informação sob a forma de fotos ou vídeos, embora não possuam funcionalidades específicas para o acesso e organização dos dados.

Outro tipo de aplicações que pode ser classificado como de partilha de dados, são os sistemas de apoio ao ensino que permitem a gestão e manipulação de informação

de diferentes tipos de *media* tais como vídeos, apresentações, fotos e textos. Paralelamente, estes sistemas disponibilizam também funcionalidades para a interacção entre utilizadores e para especificação de diferentes formas de organização que reflictam as relações específicas existentes nestes ambientes.

Como exemplos destas aplicações podem ser referidos o ClassRoom Presenter [AAMS05], aplicação específica para TabletPCs que permite aos alunos efectuarem anotações em tempo real sobre os *slides* apresentados durante a aula, e ainda a troca e partilha de documentação (por exemplo, testes e respostas) entre o professor e os alunos.

O sistema DyKnow [Ber06, Sof07] é uma aplicação também desenvolvida para TabletPCs, que permite não só a partilha de documentação mas também o trabalho colaborativo entre grupos de trabalho. Possibilita ainda a monitorização em tempo real, por parte do professor, do trabalho que está a ser realizado pelos alunos, sendo possível o estabelecimento de interacção directa com um membro ou com toda a classe em simultâneo.

Outro exemplo de aplicação especialmente concebida para ensino é o VideoStore [CC07]. Trata-se de uma aplicação Web que disponibiliza um conjunto de materiais escolares, tais como vídeos, apresentações e textos, fornecidos pelo professor, que podem ser consultados pelos alunos das diferentes turmas, os quais podem deixar as suas anotações. O professor pode ter acesso a um conjunto de ferramentas que lhe permitem, por exemplo, analisar quais os documentos e tempo do vídeo que foram mais consultados pelo alunos.

Ainda outro tipo de aplicação, estas com um carácter mais relacionado com a colaboração entre utilizadores, são os *wikis* e *weblogs* ou mesmo os fóruns de discussão que permitem a definição de grupos de utilizadores com privilégios especiais (grupos) para a produção e modificação de conteúdos.

Um *wiki* permite a edição colectiva de documentos por parte de utilizadores registados permitindo que este colaborem na criação de um *website* através da colocação e edição de conteúdos [LC01, TCC04]. Sendo um dos *wikis* mais conhecidos a Wikipedia [wik07d], é uma enciclopédia *online* que é construída através da contribuição activa de diversos participantes. De um modo geral, os *wikis* são utilizados como uma forma de disponibilizar informação de um modo simples em Intranets. Foi com essa finalidade que surgiu o primeiro wiki (WikiWikiWeb) em 1994, desenvolvido por Ward Cunningham[11]. Existem muitos exemplos de *wikis*[12], que podem ser configurados de acordo com as preferências e funcionalidades pretendidas pelos seus utilizadores, tais como WikiSpaces (*www.wikispaces.com*), wetpaint (*www.wetpaint.com*), GeboGebo

[11] Este *wiki* pode ser acedido no endereço *http://c2.com/cgi/wiki?WelcomeVisitors.*

[12] No *site* www.wikimatrix.org [Tea07] pode ser visualizada uma lista com muitos dos *wikis* e dos *weblogs* actualmente existentes.

(*www.gebogebo.org*) ou MediaWiki (*www.mediawiki.org*). Todos estes sistemas pressupõem que inicialmente o utilizador tem que efectuar o seu registo e configurar um conjunto de características próprias, podendo depois proceder à definição dos seus *wikis* propriamente ditos, onde pode então definir a estrutura e lista de permissões.

Os *weblogs*, mais vulgarmente conhecido por *blogs*, são aplicações que permitem disponibilizar através de uma página Web diferentes tipos de materiais tais como texto[13], apresentações, imagens ou vídeo. A maior diferença relativamente aos *wikis* deve-se ao facto de um *blog* ser essencialmente mantido por um indivíduo, onde outros utilizadores podem deixar os seus comentários, sendo de um modo geral mais "abertos" em termos de acessibilidade que um *wiki*. Actualmente os *blogs* constituem uma forma de apresentar artigos de opinião sobre os quais os leitores podem deixar os seus próprios comentários de um modo interactivo e simples. Como exemplo de software que serve de suporte ao desenvolvimento de *blogs* podem ser referidos, de entre um vasto número[14], o blogger da Google (*http://www.blogger.com*), o Wordpress da Wordpress Development Team (*http://wordpress.org*) e o b2evolution da evoTeam (*http://b2evolution.net/*).

2.3.2 Características e funcionalidades típicas

Nas diversas aplicações de partilha de informação existentes, algumas das quais foram anteriormente mencionadas, podem ser observadas algumas funcionalidades e comportamentos que permitem caracterizar a forma como estas efectuam a gestão da informação e o acesso à mesma:

- Estrutura da informação e tipo de *media*: a forma como os dados são armazenados depende não só do tipo de dados (texto, imagem, áudio ou vídeo) manipulados como também da forma como estes são consultados, o que condiciona a estrutura do suporte de dados. Outros factores como, por exemplo, o número de utilizadores, a dimensão dos dados manipulados ou tempo de acesso, são condicionantes da forma como é estruturada e armazenada a informação. Nos casos mais simples, pode optar-se por um sistema de ficheiros, ou em casos onde existem maiores condicionantes, optar-se por sistemas de base de dados que permitem oferecer parâmetros de segurança mais elevados. O acesso à informação pode ser efectuado através de uma simples página ou através de interfaces mais especializadas para o efeito.

- Consulta/pesquisa da informação

[13]Os conteúdos (entradas) são apresentados de acordo com a sua data de entrada, assim como diversos comentários.

[14]Pode ser consultada uma lista de *blogs* existentes em *www.weblogmatrix.org*

  - com base em meta-informação associada aos dados, onde pode ser indicado um conjunto de propriedades que têm que ser observadas;
  - com base em analogias, onde podem ser identificados padrões de características nos dados a pesquisar;
  - por explicitação directa dos dados que se pretende consultar (identificador).

- Actualização da informação
  - perfil e características do utilizador, possibilidade dos utilizadores modificarem os dados fornecidos no seu registo;
  - dados (textos, fotos, audio ou vídeo), que podem ficar acessíveis para todos os participantes, para um grupo restrito de utilizadores com permissões ou somente para o produtor.

- Coordenação
  - sincronização
    - assíncrona (*offline*), é assumida uma utilização indirecta do tipo repositório de dados, onde a informação é colocada no sistema para posterior consulta, não existindo nenhum outro tipo de "contacto"/sincronização entre os produtores e os consumidores da informação;
    - síncrona (*online*), associada normalmente ao trabalho colaborativo e onde o material produzido tem que ser observado com uma aparência de simultaneidade por todos os participantes.
  - notificação: a colocação de informação no sistema, ou a sua modificação, pode desencadear um processo de notificação dos membros "interessados". A notificação pode ser efectuada através do envio de uma mensagem (por exemplo, por *email*) ou pela disseminação de eventos.

2.4 Interacção em contextos móveis

Outra das áreas que tem vindo a sofrer grande desenvolvimento, quer nos dispositivos quer na generalização da sua utilização, é a área das comunicações móveis. Os dispositivos possuem cada vez maiores capacidades computacionais e as infraestruturas de redes móveis (*wireless*) também se encontram cada vez mais disseminadas e possuem uma maior fiabilidade. Este conjunto de factores leva a que um cada vez maior número de utilizadores tenham a possibilidade de estarem ligados entre si, em qualquer lugar. Como seria de esperar, estas novas capacidades tecnológicas motivaram o surgimento de uma nova classe de aplicações que tiram partido não só das acrescidas capacidades de conexão e de computação mas também da mobilidade

que permitiu ao seus utilizadores. Dentro desta área de desenvolvimento existem diversos conceitos que têm vindo a ser definidos e repensados, incluindo conceitos como o de computação móvel, contexto e ubiquidade.

Os computadores passaram a integrar objectos do nosso dia a dia e cada vez mais se pode dizer que eles passaram a deslocar-se connosco para qualquer lugar. A utilização de dispositivos computacionais, quando em movimento, generalizou-se e conceitos como o de computação móvel assumem uma especial relevância. A computação móvel tem que endereçar diversas questões [FZ94]:

- comunicações: implicações de utilização de redes *wireless*, mais susceptíveis às quebras de conectividade, menor largura de banda, e condições de rede mais variáveis
- mobilidade: alterações na localização, com consequências na alteração dinâmica de endereços, dependência da localização relativamente a serviços oferecidos e respostas a questões dos utilizadores e alterações de configuração, assim como perda de qualidade de serviços ao deslocarem-se no espaço (afastando-se dos emissores)
- portabilidade: dispositivos fáceis de transportar e de manusear, questões de usabilidade e de interface com o utilizador, maior probabilidade de perda de informação.

As soluções que têm surgido não são genéricas e dependem da aplicação, do local onde vai ser utilizada, dos dispositivos e também do tipo de utilizadores a que se destina.
A mobilidade endereça problemas relacionados com a possibilidade de utilizar a tecnologia computacional mantendo a conectividade. De um modo geral, para suporte da conexão estamos a assumir a ligação sem fio ("wireless") através de um dos protocolos de rede existentes (Wi-Fi 802.11, bluetooth ou infravermelhos). Actualmente, dispositivos como os *laptops*, PDAs ou telemóveis são dispositivos que garantem aos seus utilizadores capacidades de conexão com um certo grau de fiabilidade. Este tipo de dispositivos continua ainda a possuir certas restrições a nível tecnológico, tais como a sua capacidade de computação de comunicação e principalmente de bateria e de interface, sendo por isso ainda a sua utilização um pouco condicionada por estes factores. No entanto, nos últimos tempos tem vindo a observar-se uma melhoria significativa destes dispositivos principalmente ao nível das capacidades computacionais e de conexão. Daí o surgimento de diversas aplicações que tiram partido das características particulares destes elementos, em particular na forma como a informação é disponibilizada aos utilizadores e na forma como estes interagem, quer entre si, quer com o meio envolvente.

Esta possibilidade está associada ao conceito de contexto, que assume especial importância quando em ambientes móveis. A caracterização do conceito de contexto, segundo diversos autores [SAW94, Pas98], abrange um conjunto de circunstâncias envolventes, tais como localização, identidade dos elementos que se encontram na proximidade, objectos, assim como um conjunto de factores do meio ambiente como por exemplo, hora do dia, temperatura, luminosidade e nível de ruído. A caracterização do meio ambiente envolvente assume especial importância, dado que as suas alterações podem condicionar o comportamento deste tipo de aplicações. Procurando uma definição mais abrangente [DA00], contexto pode ser:

> "Qualquer informação que pode ser utilizada para caracterizar a situação de determinada entidade (pessoa, local ou objecto) que seja considerada relevante para a interacção entre o utilizador e a aplicação, incluindo o utilizador e a aplicação. Tipicamente o contexto está associado a uma localização, identidade e estado da pessoa, grupo e objectos computacionais e físicos."

De certa forma associado ao conceito de contexto, surge a noção de ubiquidade que se refere à forma como os computadores podem integrar o meio envolvente de forma não intrusiva, passando a fazer parte integrante deste, tornando-se praticamente "invisíveis" para o utilizador e assumindo a forma de artefactos utilizados no nosso dia a dia [Wei99]. Este termo foi apresentado pela primeira vez em 1991 por Mark Weiser, que argumentava que os recursos de computação seriam omnipresentes na vida diária e seriam interligados com o fim de fornecer a informação ou serviços que os utilizadores requerem em qualquer lugar. A ideia subjacente é que os computadores deixam de ser postos de trabalho fixos e passam a estar integrados nos mais variados objectos de uso comum, de forma perfeitamente transparente para o utilizador. Esta noção pode aplicar-se como forma de aumentar as potencialidades de trabalho ao nível profissional e também ter uma cada vez maior influência na nossa vida quotidiana.

2.4.1 Sistemas e aplicações

As aplicações para interacção entre múltiplos utilizadores, desenvolvidas para contextos móveis, podem ser divididas em dois grandes grupos:

- aplicações que têm por base adaptações para dispositivos móveis, de aplicações Web já existentes;
- aplicações especialmente concebidas para os dispositivos móveis.

No primeiro caso, existem diversos exemplos, como o YouTube, que permite aos utilizadores fazerem o carregamento dos seus vídeos directamente dos seus dispositivos móveis, o Google Mobile que disponibiliza o acesso a alguns dos diversos recursos

da Google através de um dispositivo móvel[15] tais como o Gmail, a busca de sites e imagens e o Google Maps. Aplicações como por exemplo, o Facebook mobile, oferecem, para além de uma versão especialmente desenhada para ser visualizada em telemóveis, a possibilidade de fazer o carregamento de fotos e anotações e ainda enviar e receber mensagens de texto de grupos de amigos.

O sistema Twitter (*www.twitter.com*) pode também ser considerado dentro desta linha de aplicações[16] e permite aos utilizadores enviarem pequenas mensagens de texto para um grupo de amigos pré-definido. O "contacto" destes pode ser um endereço de *email* ou de mensagens ou mesmo um número de telemóvel e as mensagens podem ser recebidas por todos os que se inscreveram para recebê-las, sendo o objectivo principal a manutenção constante de contacto entre os membros do grupo de amigos.

Paralelamente têm também vindo a ser desenvolvidos serviços que permitem a distribuição e acesso a informação dependente da localização, como por exemplo o Dodgeball [Hum07], que permite efectuar a disseminação de mensagens de texto aos grupos de amigos (ou amigos de amigos) mas somente os que se encontram na proximidade do lugar onde o utilizador se encontra actualmente[17]. Neste caso não se necessita de sensores de posicionamento (tipo GPS), sendo da responsabilidade do utilizador enviar uma mensagem ao sistema Dodgeball com a sua posição actual.

No segundo grupo de aplicações referido, existe uma grande diversidade, sendo algumas das mais representativas aplicadas ao domínio do turismo e entretenimento.

Alguns exemplos de aplicações como o CiberGuide [AAH+97, web07b] permitem, tendo por base a localização actual do utilizador, oferecer informação e serviços que melhor lhe servem, tais como, informação sobre os pontos de interesse turístico que se encontram na proximidade, responder a simples perguntas sobre esses mesmos pontos de interesse e ainda deixar notas ou difundir mensagens para o grupo de turistas a que pertence.

Dentro desta mesma linha de desenvolvimento surgiu ainda o GUIDE [CDM+00, DCME01] desenvolvido na universidade de Lancaster e especificamente desenvolvido para ser utilizado por visitantes daquela cidade. O sistema é suportado por uma infraestrutura de comunicações (WiFi - protocolo IEEE 802.11) que foi especialmente implantada para este fim e a informação diponibilizada aos utilizadores depende do contexto pessoal (características/preferências do utilizador) e do ambiente (hora do dia e localização).

Um projecto interessante dentro desta linha é o MobiLife (*www.ist-mobilife.org*), que tem como objectivo principal desenvolver um conjunto de aplicações e serviços que

[15] O Google Mobile, pode ser visualizado a partir de qualquer dispositivo que tenha uma ligação com a internet e um browser WAP.

[16] Este tipo de aplicações tem vindo a ser denominada de MoSoSo - Mobile Social Software.

[17] Este trata-se de um serviço pago e actualmente só disponível em algumas cidades dos Estados Unidos.

possam ser utilizados na rotina diária de um grande número de utilizadores. O sistema oferece a informação personalizada de acordo com o contexto pessoal, de grupo e de ambiente. Ao nível do contexto pessoal entra em linha de conta com informação referente às características próprias do utilizador e dispositivo, às suas preferências e ao seu histórico de acções. O contexto de grupo considera relevante a informação que permita captar a dinâmica de grupo e do seu comportamento, sendo definido por um conjunto de características próprias e de preferências. Ao nível do contexto do ambiente, pode existir por exemplo, informação sobre o local onde o utilizador se encontra actualmente, o seu posicionamento geográfico e a hora do dia. Esta informação pode condicionar a forma como a informação é apresentada ao utilizador. Por exemplo, se este se encontra dentro do carro as mensagens que lhe chegam não devem ser apresentadas sob a forma de texto e sim como discurso falado [KBB+06].

Existem ainda diversas aplicações mais localizadas, restringidas a um edifício por exemplo, como é o caso da aplicação desenvolvida para o museu Marble em Itália [MPSS06, SPRL07], que disponibiliza em PDAs (colocados à disposição dos utilizadores) um conjunto de informações sob a forma de fotos, vídeos e texto, referentes às obras em exposição. O mecanismo de localização utilizado tem por base um sistema de infra-vermelhos, que lhe permite detectar em que sala o utilizador se encontra e, com base nisso, apresentar o respectivo mapa informativo, sendo os objectos expostos detectados através de RFID (*Radio Frequency Identification*).

Outro projecto, que desenvolveu aplicações no domínio das aplicações para turismo e desenhadas para um espaço especifico, é o InStory [CAC+05]. Foi desenvolvido um conjunto de aplicações que permitem ao visitante do espaço obter informação de acordo com as suas preferências e localização actual. Algumas das aplicações desenvolvidas têm por base um pequeno jogo, onde é pedido ao utilizador que responda a determinadas questões ou encontre determinado local ou objecto, sendo o objectivo principal levar o visitante a percorrer todo o espaço e conhecer um pouco a sua história. Outra das aplicações desenvolvidas permite ao visitante ir recolhendo fotos dos diferentes pontos, podendo também ter acesso a fotos desses mesmos locais obtidas por anteriores visitantes [DJFC07, RMJDFC07].

2.4.2 Funcionalidades típicas

Das diversas aplicações analisadas observou-se que, de um modo geral, as funcionalidades pretendidas no domínio das interacções móveis são muito idênticas às encontradas nas aplicações onde não existe mobilidade física. As diferenças surgem pelo facto de as condições do meio onde o utilizador/dispositivo se encontra num dado momento condicionarem o comportamento da aplicação e a forma como esta reage às modificações do contexto, para além dos próprios condicionalismos inerentes às li-

mitações dos próprios dispositivos, tais como as suas capacidades computacional a tempo de actividade (bateria) e conectividade. Os factores do meio a considerar estão intimamente relacionados com a aplicação. No entanto, os mais relevantes a considerar são [DA00] habitualmente:

- Identidade, este tipo de factor está relacionado com a possibilidade de identificação única de cada indivíduo;

- Localização, factor que determina qual a posição do utilizador/dispositivo em cada instante. Com base nesta informação pode ser extraída outra informação como, por exemplo, a relativa aos indivíduos que se encontram na proximidade. A obtenção da informação da localização do utilizador pode ser adquirida por informação explícita deste ou automaticamente através do recurso a dispositivos de localização, como por exemplo GPS (para espaços exteriores) ou RFID (para edifícios). A localização pode conter informação mais detalhada podendo, por exemplo, conter informação informação referente à orientação actual do utilizador e elevação do terreno. A informação referente à localização pode também ter um caracter absoluto (latitude e longitude) ou relativo (distância a que o utilizador se encontra de um determinado local). A partir da localização, outro tipo de informação de contexto pode ser deduzida como por exemplo, quem ou o que se encontra na vizinhança do local;

- Actividade, este factor está associado ao conjunto de características intrínsecas observadas que podem ser consideradas. Quando num local, podem referir-se por exemplo, a uma temperatura, humidade ou nível de ruído. Quando referente a um indivíduo, pode ser por exemplo, a actividade que está a ser efectuada por este, como por exemplo se está a escrever ou se está a falar. O factor actividade pode também caracterizar o comportamento e estado de um grupo de indivíduos. Também a actividade, tal como a localização, pode ser utilizada como forma de deduzir mais informação;

- Tempo, é também informação de contexto dado que permite caracterizar uma determinada situação, sendo de um modo geral utilizada como forma de complementar outro tipo de informação de contexto existente.

2.5 Requisitos e dimensões para o suporte a grupos

Ao fazer a análise das diversas aplicações, anteriormente referidas, existentes no domínio da interacção e partilha de informação entre grupos de utilizadores, observou-se que existem diversas características comuns. Algumas dessas características são enunciadas de seguida:

- interacção através da troca directa de mensagens (texto, som ou imagens) entre utilizadores;
- associação de utilizadores de acordo com as suas preferências e/ou interesses específicos. Estas associações são, de um modo geral, efectuadas de forma explícita pelo utilizador, embora possam surgir sugestões por parte dos sistemas;
- definição de grupos explícitos de "amigos" por parte dos utilizadores;
- divulgação/partilha de dados (por exemplo fotos, vídeos ou textos) por diferentes grupos;
- permissão de diferentes tipos de configurações de acesso para os diferentes membros de diferentes grupos definidos pelo utilizador;

Do que foi referido anteriormente, resulta um conjunto de necessidades e problemas que se colocam a estes diferentes tipos de aplicações, em particular se se pretender dotar as aplicações de mobilidade, de um maior dinamismo nas suas configurações e interacções entre participantes e de uma maior adaptabilidade relativamente ao estado do ambiente envolvente. Algumas das dimensões que têm que ser consideradas ao desenvolver aplicações com suporte à computação social móvel são:

- Concorrência: este factor prende-se principalmente com o facto dos acontecimentos, provenientes dos diversos participantes do sistema, poderem ocorrer simultaneamente e terem que ser tratados no momento em que ocorrem;
- Entidades distribuídas móveis: esta dimensão está principalmente associada à mobilidade dos utilizadores (e dos seus dispositivos), tendo que manter a capacidade de conexão de modo transparente para o utilizador, e também fazendo variar o estado da aplicação de acordo com a sua movimentação/localização;
- Volatilidade: ao nível da volatilidade há que considerar não só a volatilidade dos dados como também dos utilizadores, podendo estes entrar e sair dinamicamente do sistema e das estruturas sociais a que pertence. Esta noção de volatilidade está associada a factores de ordem temporal e também espacial;
- Formas de interacção: as interacções entre os diversos elementos podem ser de diferentes tipos como, por exemplo, directa entre dois utilizadores (*peer-to-peer*), de um utilizador para um grupo de outros (disseminação), ou ainda através da partilha e colocação de informação no repositório comum a que podem posteriormente aceder;
- Natureza dinâmica dos problemas: o carácter não determinístico com que são estabelecidas as comunicações entre os diversos participantes, como estes podem participar (entrar/sair) do sistema e dos grupos em que participa ou ainda a

ocorrência de modificações ao nível do ambiente envolvente, podem condicionar o desenrolar da execução e a forma como são estabelecidas as interacções;

- Organização dos elementos de acordo com as suas características: a possibilidade de personalizar as aplicações e a informação, de acordo com as características pessoais de cada utilizador, assim como sugerir ou mesmo agregar utilizadores de acordo com as suas preferências;

- Heterogeneidade ao nível dos dados: as aplicações têm que lidar de um modo geral com diferentes tipos de informação, provenientes de diversas origens e de diferentes *media*, como vídeos, fotos, texto ou audio;

- Diversidade de dispositivos: levanta questões relacionadas com o facto das aplicações terem que ser executadas em diferentes plataformas de suporte, o que motiva a criação de infraestruturas genéricas. Estas devem ter em atenção os problemas dos diferentes sistemas de suporte e também as diferentes interfaces de utilizador que têm que ter em conta os diversos tipos de dispositivos alvo.

2.6 Conclusão

Ao longo deste capítulo efectuou-se um levantamento de diversas aplicações/serviços que permitem o estabelecimento de interacções entre múltiplos utilizadores e a definição de formas de organização e estruturação destes e da sua informação.

Foram identificados um conjunto de necessidades e requisitos, que as diversas aplicações e serviços analisadas necessitam, com o objectivo de identificar os possíveis modelos de comunicação existentes que permitem caracterizar os padrões de interacção entre entidades e também estruturar a forma como estas se relacionam. Esses modelos, que são analisados no capítulo 3, permitem dar resposta a algumas das necessidades e requisitos descritos, embora não de uma forma completa, pelo que no capítulo 4 é apresentada a proposta de um modelo cujo objectivo é captar e integrar diversas das funcionalidades referidas de forma a que a construção deste tipo de aplicações se torne mais simples.

3

Modelos de comunicação: grupos, espaço partilhado e eventos

Conteúdo

Neste capítulo discutem-se modelos de comunicação que oferecem funcionalidades para a interacção entre múltiplas entidades computacionais e a sua estruturação. São referidos, em particular, os modelos de comunicação baseados em grupos, espaços partilhados e eventos, sendo analisadas as suas funcionalidades que permitem suportar alguns dos serviços referidos nas aplicações descritas no capítulo anterior.

3.1 Introdução

Das diversas aplicações analisadas ao longo do capítulo anterior, observou-se que estas possuem, de um modo geral, necessidades de estruturação, partilha de informação e estabelecimento de interacções entre as entidade participantes.

As aplicações consideradas exibem na sua maioria um comportamento dinâmico e são constituídas por componentes distribuídas, beneficiando por esse motivo de formas de estruturação baseadas em grupos de entidades. Um modelo de organização baseado em grupos de entidades, permite capturar essa característica bem como definir formas de interacção estruturada entre grupos de elementos.

Um problema central na modelação de aplicações interactivas distribuídas é o seu requisito quanto à partilha de informação. Uma dimensão deste problema relaciona-se com o acesso a repositórios ou sistema de informação persistentes, que registam elementos sobre as entidades e os seus atributos/perfis, bem como sobre os conteúdos por elas manipulados. Outra dimensão da partilha de informação ocorre associada às interacções que múltiplas entidades estabelecem durante a execução de uma aplicação, e que permite que estas estabeleçam formas de colaboração e cooperação entre si.

Ao nível da diversidade de modelos de interacção existentes e por forma a capturar a diversidade de interacções entre entidades, que caracterizam as aplicações anteriormente referidas, podem considerar-se as seguintes principais dimensões:

1. semântica da interacção, que pode ser síncrona ou assíncrona;

2. estabelecimento de interacção, de forma directa ou indirecta;

3. modelo de comunicação, por memória partilhada ou distribuída.

No que diz respeito à semântica das interacções, ambas as semânticas referidas são importantes para as aplicações consideradas. Em particular, dado o caracter reactivo exibido pelas aplicações e a sua natureza dinâmica na resposta a modificações do meio envolvente, as diversas entidades participantes devem ter a capacidade de reagirem a eventos, que lhes são notificados de forma assíncrona.

Quanto à forma como se processa o estabelecimento de interacções, verificam-se tanto necessidades de padrões de interacção directa, em que duas ou mais entidades interagem entre si, existindo nesta situação uma forte ligação entre os intervenientes, como padrões de interacção indirecta, existindo nesta situação mecanismos baseados em difusão e em meios de comunicação intermédios, como por exemplo filas globais de mensagens ou espaços de memória partilhada.

Ao nível dos modelos de comunicação, as duas grandes classes a considerar são a memória partilhada e a memória distribuída. Estas duas formas devem ser contempladas, pois elas decorrem naturalmente do carácter das aplicações analisadas. No caso da memória distribuída, as duas principais formas de comunicação baseiam-se

em troca de mensagens ou em invocação de métodos remotos e são adequadas para suportar a interacção entre entidades distribuídas. No caso da memória partilhada, a comunicação entre entidades é efectuada através de uma abstracção de um espaço comum, sobre o qual podem efectuar consultas (leituras) e actualizações (escritas), podendo este espaço suportar, não apenas comunicação e partilha de informação, mas também a coordenação de entidades.

Neste trabalho, desenvolveu-se um modelo que combina diversos mecanismos característicos das dimensões acima apontadas. Em particular, consideram-se:

- os grupos de entidades, como unidades de estruturação das interacções;
- um espaço partilhado, confinado a cada grupo, permitindo acesso a informação exclusivamente partilhada pelos membros do grupo e possibilitando também a coordenação entre eles;
- as mensagens, como forma de comunicação directa entre as entidades;
- os eventos, como forma de notificação assíncrona das modificações dinâmicas que ocorrem.

Como objectivo de melhor caracterizar esta problemática, discutem-se de seguida diferentes modelos que apresentam funcionalidades como as acima mencionadas. É inicialmente apresentado o modelo de comunicação em grupo e as suas principais características e propriedades. De seguida, é apresentado um modelo de espaços partilhado, sendo dado especial relevo ao modelo Linda [CG89], que é o trabalho mais representativo desta linha de investigação. Finalmente é analisado o modelo de comunicação baseado em eventos, que permite uma forma de interacção assíncrona entre elementos.

3.2 Modelos de comunicação em grupo

Nesta secção resumem-se as principais características e componentes dos sistemas de comunicação em grupo, sendo referidos alguns dos mais relevantes.

Os primeiros sistemas de comunicação em grupo surgiram nos anos 80, com o objectivo de resolver alguns dos problemas que ocorriam na gestão de dados distribuídos, em particular nas bases de dados. Problemas relacionados com a replicação de informação e com a coordenação de múltiplas cópias, motivaram o aparecimento de sistemas que garantissem fiabilidade na comunicação em grupos de processos distribuídos [CKV01].

A abstracção do modelo de comunicação em grupo possibilita, de uma forma simples, a organização de grupos no contexto dos quais se processa a difusão de mensagens. São oferecidas diferentes semânticas quanto às garantias relativamente à entrega

de mensagens, podendo ser assumido que uma mensagem dirigida ao grupo é entregue a todos os membros filiados do grupo que se encontrem activos. Estes sistemas de comunicação assentam sobre uma camada intermédia que realiza um conjunto de protocolos e que permite abstrair das dificuldades e/ou possíveis falhas na comunicação entre entidades distribuídas. Possibilita-se, assim, a simulação de um ambiente que cria a ilusão de uma rede ideal, sem perdas de mensagens e com garantias de ordenação de entrega das mesmas; as falhas, uma vez detectadas, são "vistas simultaneamente" (ou de forma coordenada) por todos os membros do grupo [Kei01].

Abstracções como estas são alguns dos motivos pelos quais a agregação de diversas entidades em grupos pode facilitar o desenvolvimento de aplicações distribuídas, em particular se se pretende desenvolver aplicações tolerantes a falhas. Um sistema de comunicação em grupo difere dos sistemas simples de troca de mensagens devido às primitivas de comunicação que suporta e às garantias que são oferecidas pelo modelo [CKV01]. Tipicamente estes são sistemas assíncronos, nos quais não existe um limite temporal definido para a entrega de mensagens ou para a execução de dois passos consecutivos de execução [Vit98]. Por isso uma das mais importantes garantias é a garantia de consistência, que é obtida através da coordenação entre a troca de mensagens e as modificações da constituição do grupo em cada instante.

A constituição de um grupo pode sofrer alterações ao longo do tempo durante a execução da aplicação, já que novas entidades[1] podem juntar-se ao grupo e outras podem sair. Ao conjunto de membros actuais de um grupo denomina-se "visão de grupo"da filiação. A entrada ou saída de membros provoca alterações na visão da constituição, que é actualizada pelo serviço de filiações (*group membership*). Uma das garantias principais é a de difundir, para todos, a constituição actual do grupo (visão sincronizada) tendo esta que ser fiável (atómica) já que é a que vai garantir a consistência e a coordenação entre todos os filiados. Ao ter uma difusão atómica é possível garantir que uma mensagem é entregue a todos os membros do grupo ou não é entregue a nenhum [CKV01].

Na sua essência, cada grupo possui um nome global e único, que é utilizado para a comunicação entre uma qualquer entidade do sistema e o grupo. Ou seja, um elemento interage (comunica), com o grupo utilizando esse identificador. Qualquer entidade externa ao grupo pode comunicar com os membros do grupo através do envio de mensagens dirigidas ao grupo. Aqui os modelos de comunicação distinguem os "grupos abertos " dos "grupos fechados", consoante é permitida ou não, a comunicação de uma entidade externa com os membros do grupo. É da responsabilidade do serviço de comunicação de grupo garantir que a mensagem é entregue (na ordem especificada pelo protocolo escolhido), a todos os membros do grupo.

[1]Nos sistemas inicialmente propostos uma entidade era vista como um processo, ao nível do SO que se encontrava em execução. No modelo genérico, a interpretação de uma entidade pode corresponder à noção de objecto, componente ou serviço.

Um sistema de comunicação de grupo tem por base um serviço de grupos, em que conjuntos de processos ou objectos se agregam e aos quais ficam disponíveis várias primitivas para difusão de mensagens entre os membros. Na transmissão de mensagens, podem ser garantidos diferentes tipos de ordenação de eventos, de acordo com a semântica que se pretende ver implementada.

Embora existam múltiplos sistemas de comunicação de grupo, com diferentes características entre si, são comuns os seguintes tipos de serviços: filiação, difusão e sincronia virtual. Nas subsecções seguintes, é efectuada uma breve discussão destes serviços.

3.2.1 Filiação

O serviço de filiação (membership) tem como objectivo principal a manutenção de uma lista de entidades[2] correntemente activas. Este serviço tem também a responsabilidade de comunicar, para as camadas superiores, qual a constituição e estado actual do grupo[3].

Este é o serviço responsável pela criação/formação e destruição de grupos, assim como pela gestão das entradas e saídas de membros de um grupo. É também da sua responsabilidade reagir a possíveis falhas, assinalando quais as entidades com as quais não se consegue comunicar (o que pode significar uma falha do processo ou uma falha na ligação). Sempre que for detectada uma alteração na constituição do grupo, todos os actuais membros activos deverão receber uma notificação. Desencadeia-se então o processo de instalação da nova visão, ou seja um novo estado do grupo é comunicado a todos os membros activos [BBD97]. Desta forma, a cada instante, cada elemento do grupo sabe quais os outros membros que ele reconhece como activos, ou seja cada membro possui uma visão actualizada do grupo. Existem diversos protocolos de filiação, que variam pela forma como as visões são construídas e como lidam com falhas.

O modo como os sistemas reagem a falhas está intimamente associado à forma como estes gerem os seus membros. Podem ser identificados dois tipos distintos de sistemas: os de partição primária e os com múltiplas partições. Nos sistemas que implementam o protocolo de partição primária, no caso de ocorrência de falhas só são considerados como membros activos os que pertencem à partição primária [BBD97]. Esta partição foi previamente configurada ou seja a sua constituição é pré-definida. No caso de ocorrer uma falha no sistema, só é garantida a entrega de mensagens aos membros que pertencem à partição primária e da mesma forma só estes podem enviar mensagens (todos os outros membros são assumidos como falhados). Nos sistemas com múltiplas partições, ao ocorrerem falhas, podem ser definidas várias partições, dentro das quais continua a existir possibilidade de transmissão de mensagens. É pos-

[2] Aqui uma entidade é assumida como um processo.

[3] A constituição do grupo num dado instante é denominada "visão da constituição do grupo" (*view) [BBD97].*

sível a instalação de visões concorrentes disjuntas, novas visões podendo resultar da junção ou separação de partições anteriores, o que permite que seja possível continuar as operações, mesmo quando não se comunica com alguns dos membros (processos). No entanto, a recuperação em caso de falha nestes sistemas, torna-se mais complexa e com problemas de manutenção de consistência.

3.2.2 Difusão de mensagens

Esta funcionalidade é um dos serviços base dos modelos de comunicação de grupos já que é a responsável pela comunicação dentro de um grupo, ou seja pela difusão e entrega de mensagens dentro de um grupo. Este serviço deve obedecer a um conjunto de requisitos de modo a garantir a fiabilidade das comunicações no grupo [CDK00]:

- *Validade*: se um membro activo difunde uma mensagem m, então eventualmente esta será entregue;
- *Consenso (agreement)*: se um membro activo entrega uma mensagem m, então todos os processos activos recebem m;
- *Integridade*: para todas as mensagens m, cada membro entrega m no máximo uma e vez e só se m tiver sido anteriormente difundida por um membro.

Tipicamente, a entrega de uma mensagem a um qualquer membro do grupo obedece a duas fases. Inicialmente a mensagem é recebida pelo processo e colocada numa fila, não sendo directamente entregue aos processos da aplicação. Somente após ver garantidos os critérios de fiabilidade exigidos pelo protocolo de difusão é que a mensagem é efectivamente entregue à aplicação.

Os sistemas de comunicação de grupo oferecem às camadas superiores (de aplicação) diferentes serviços de difusão de mensagem, com diferentes semânticas relativamente à ordem de entrega. Embora existam múltiplas variações, os critérios de ordenação mais frequentes são [CDK00]:

- *FIFO (first in first out) fiável*: se um processo difunde uma mensagem m antes de m', nenhum processo entrega m' sem antes ter entregado m;
- *Ordem causal*: se a mensagem m precede causalmente a mensagem m', nenhum processo entrega m' sem antes ter entregado m;
- *Ordem total*: se um processo entrega a mensagem m antes de m', nenhum outro processo pode entregar m após ter entregado m'.

Os dois primeiros critérios de ordenação garantem que a resposta a uma mensagem nunca é entregue antes da mensagem ter chegado ao seu destino. A definição de ordenação total é um pouco ortogonal à ordenação FIFO e causal, podendo por isso

ser combinada com ambas, garantindo que as mensagens são entregues pela mesma ordem a todos os membros do grupo, por forma a assegurar a consistência na replicação.

A ordenação total requer uma implementação mais pesada do que as outras, já que necessita de uma coordenação mais forte na entrega das mensagens. No entanto, é bastante útil na implementação de serviços de replicação, que exigem que as réplicas observem a mesma sucessão de actualizações Este critério é também aplicado para garantir a difusão atómica, o que tem como efeito que estas operações de difusão aparentem ser síncronas (ocorrer em simultâneo), não se entrelaçando com a entrega de outras mensagens.

3.2.3 Sincronia virtual

A sincronia virtual é um modelo que combina a noção de atomicidade com a de restrições na ordem de entrega de eventos. O principal objectivo é o de preservar a ilusão de que os eventos ocorrem instantaneamente, numa noção de tempo virtual. O modelo de sincronia virtual foi pela primeira vez proposto no sistema Isis [BJ87] e foi inicialmente apresentado no contexto de um sistema de partição primária [BBD97]. Por esse motivo o termo sincronia virtual está muito ligado a sistemas de partição primária, pelo que certos autores preferem utilizar o termo "visão sincronizada". O termo visão está directamente relacionado com o problema da filiação no grupo. Sempre que ocorrem alterações na constituição do grupo, ou seja quando ocorrem entradas ou saídas de membros do grupo, a visão deve ser actualizada e difundida para todos os membros activos do grupo. É da responsabilidade do serviço de "sincronia virtual" a gestão das visões do grupo, de modo a garantir que todos os membros do grupo possuam uma mesma visão da constituição do grupo (garantia de consistência).

Num sistema com sincronia virtual, é garantido que os eventos são observados pela mesma ordem por todos os membros do grupo. Se não ocorrem alterações na constituição do grupo, as mensagens deverão ser entregues a todos os membros que pertencem à visão actual do emissor. Se ocorrem alterações durante a entrega, deverá ser instalada uma nova visão; no entanto todas as mensagem enviadas na visão anterior deverão ser entregues aos membros ainda activos na visão anterior, antes da nova ser instalada. A sincronia virtual pode ser descrita, sucintamente, como a garantia de uma ordem total na entrega de mensagens e nas alterações da visão de grupo, permitindo aos processos tratar cada mensagem num contexto comum de filiação num grupo [Bir93].

A visão sincronizada pode ser definida da seguinte forma [Per02]:

> "se um processo instala duas visões consecutivas v_i e v_{i+1} e entrega a mensagem m na visão v_i, então todos os restantes processos que instalem ambas as visões v_i e v_{i+1}, entregam m na visão v_i"

Pode garantir-se assim que, entre duas visões consecutivas, os processos (elementos) pertencentes às duas visões, entregam o mesmo conjunto de mensagens.

Resumo de características

Uma das grandes vantagens dos modelos de comunicação em grupo é o de permitir esconder a complexidade devida a falhas, já que estas podem ser vistas como alterações nos membros constituintes do grupo (filiação), através das visões que são partilhadas por todos os membros activos do grupo. Oferecem também garantias relativas ao conjunto de mensagens entregues globalmente, função das alterações de visão que um processo observa localmente, sendo por isso possível aos membros de um grupo terem uma visão global baseada somente na sua informação local [BDM01].

Resumindo, algumas das vantagens de utilização de comunicação em grupo prendem-se com o facto de:

- o grupo poder ser encarado como uma entidade do sistema com uma interface bem definida, que pode ser acedida pelos elementos externos ao grupo, permitindo uma estruturação na organização das entidades presentes no sistema;
- o grupo permite lidar com as características dinâmicas associadas aos sistemas distribuídos, sendo as entradas e saídas de membros geridas de forma transparente para as camadas superiores;
- os grupos dão garantias de consistência e fiabilidade nas comunicações;
- os grupos permitem formas distintas de comunicação entre os seus membros, nomeadamente com mecanismos de comunicação ponto a ponto, difusão e estado partilhado (este aspecto foi particularmente explorado neste trabalho).

3.2.4 Algumas propostas de sistemas de comunicação em grupo

As primeiras arquitecturas de sistemas propostos correspondiam a sistemas monolíticos [MSW03], como é o caso do Isis, um dos primeiros sistemas de comunicação em grupo, desenvolvido na Universidade de Cornell na década de 80 [BC91]. Um dos motivos do surgimento desta proposta prendeu-se com o facto de existir uma cada vez maior necessidade de garantir a fiabilidade dos sistemas distribuídos, tornando-os mais robustos à ocorrência de falhas. Uma das formas de minimizar os problemas da ocorrência de falhas consistiu em replicar componentes, deste modo sendo possível oferecer maiores garantias de fiabilidade. No entanto, criou-se o problema de gerir e manter consistente a informação nas diversas réplicas, daí ter surgido a comunicação em grupos como forma de oferecer mecanismos de comunicação que oferecessem garantias de consistência da informação difundida aos elementos de um grupo.

No seguimento da linha da proposta do Isis, surgiram diversos desenvolvimentos com o objectivo, não só de optimizar os serviços, como também de permitir uma

melhor adaptabilidade do sistema de acordo com as necessidades da aplicação. Os seguintes são exemplos de sistemas, entre muitos outros, que foram entretanto propostos: Horus [RBG+95], Transis [ADKM92, DM96],Totem [AMMS+95, MMSA+96], Phoenix [MSW03], RMP [Bar98], Ensemble [Hay98, RBD01].

Mais recentemente, surgiram algumas implementações de sistemas baseados em Java, tendo como principal objectivo resolver problemas de portabilidade e modularidade. Por exemplo o JavaGroups [Ban98b] desenvolvido por Bela Ban, que se trata de uma readaptação do sistema Ensemble e Horus, que foi desenvolvido em ML, para Java. Também na mesma linha de investigação, surgiu o sistema Jgroup [HMM05], desenvolvido na Universidade de Bolonha, no qual são considerados os problemas de gestão de consistência em grupos de objectos replicados.

Surgindo no seguimento dos trabalhos anteriormente desenvolvidos na Universidade de Cornell, o sistema JavaGroups baseia-se numa plataforma de comunicação desenvolvida em Java que oferece uma infraestrutura de suporte à comunicação em grupo com garantias de fiabilidade na entrega de mensagens [Ban98a]. No caso de sse utilizar ordenação total, possui um conjunto de funcionalidades que lhe permitem garantir que cada membro do grupo recebe a mesma sequência de mensagens na mesma ordem. Um dos conceitos centrais na arquitectura do sistema JavaGroups é o conceito de "canal", que pode ser visto como o elemento que representa o grupo. Cada "canal" possui um nome próprio que o identifica, vários "canais" com o mesmo nome formando um grupo de processos [Ban98a]. Os "canais" possuem um interface com as funcionalidades básicas de interacção com o grupo: notificação de entrada e saída de membros; envio e recepção de mensagens; acesso a constituição de um grupo (membros). Os clientes podem ligar-se a um "canal", especificando o nome do grupo ao qual pretendem filiar-se, e através do qual podem trocar mensagens e receber notificações de alteração na constituição do grupo. Os "canais" com o mesmo nome são reconhecidos entre si, sendo que uma mensagem enviada por um "canal" será recebida por todos os "canais" que possuirem o mesmo nome. As aplicações podem utilizar esta interface se pretenderem efectuar um desenvolvimento a um nível mais baixo das comunicações; no entanto o sistema JavaGroups tem disponível um conjunto de primitivas que oferecem um maior grau de abstracção, em que esquemas mais complexos de comunicação são disponibilizados, tal como por exemplo, esquemas de eleição de um novo coordenador de grupo no caso de ocorrência de falha do actual. O sistema JavaGroups possibilita, ao programador das aplicações, a escolha do nível de abstracção que melhor lhe convém, sendo esta uma plataforma que oferece uma grande modularidade. Mais recente esta plataforma evoluiu para o JGroups[4] [JGr06].

Outra plataforma de comunicação em grupo, também desenvolvida em Java, foi o Jgroup [MM00], que implementa uma extensão do modelo de objectos distribuídos,

[4]JGroups - "A Toolkit for Reliable Multicast Communication"

baseada no paradigma da comunicação de grupo [Jgr07]. Um dos objectivos principais da sua criação foi o de suportar o desenvolvimento de aplicações em sistemas distribuídos com partições de processos, tentando facilitar e simplificar a cooperação entre grupos de objectos servidores replicados e, do lado do cliente, oferecer mecanismos transparentes para invocar métodos dos objectos grupo, como se se tratassem de entidades simples não replicadas. O Jgroup é baseado em duas abstracções fundamentais: grupos de objectos remotos e objectos remotos replicados. Do ponto de vista de um servidor, um grupo de objectos remotos consiste de uma colecção de réplicas de objectos que implementam o mesmo conjunto de interfaces e coordenam a sua execução de modo a serem vistas como um objecto remoto não replicado. As réplicas que formam um grupo de objectos remotos cooperam através de um sistema de comunicação em grupo que aceita partições, cuja função é simplificar o desenvolvimento de protocolos consistentes, necessários para garantir a fiabilidade e qualidade de serviço. Neste modelo, os clientes só têm acesso directo para interacção com grupos de objectos replicados e não com cada uma das réplicas. Cada grupo implementa uma ou mais interfaces remotas, cujos métodos podem ser invocados através do mecanismo de RMI - *Remote Method Invocation*.

3.3 Modelos de espaços partilhados

Nesta secção discutem-se os principais conceitos presentes nos modelos de espaços partilhados, centrando a atenção mais particularmente nos baseados em espaços de tuplos.

3.3.1 O modelo Linda

O conceito de espaço de tuplos teve a sua origem no modelo de coordenação para processamento paralelo "Linda", proposto por David Gelernter e Nicholas Carriero, na Universidade de Yale [CG89,CG01]. Este modelo define uma abstracção de um espaço de memória partilhada para ambientes distribuídos, sem necessidade de existência de uma infraestrutura de partilha de memória física ao nível do hardware.

O modelo introduziu pela primeira vez num sistema a ideia de separação entre os conceitos de coordenação de sistemas concorrentes e a computação. A criação desta separação permite que diferentes processos de computação (em diferentes linguagens) cooperem entre si, utilizando as mesmas primitivas de acesso ao espaço partilhado.

O modelo Linda, mais do que uma linguagem, é um sistema abstracto de coordenação que oferece uma interface simples para a comunicação entre processos [CG01]. Este tipo de abordagem permite efectuar a comunicação e a coordenação entre entidades, sem necessidade destas se "conhecerem", dado que a interacção entre elas é efectuada através do espaço de tuplos, sendo por isso indirecta e anónima.

Um tuplo é uma estrutura constituída por uma lista ordenada de campos de um determinado tipo [CG89], que podem ou não encontrar-se instanciados com um valor. Exemplo de tuplo com todos os campos instanciados: $< "string", 10, 20, "string2" >$. Tuplo com alguns campos não instanciados: $< "string", -, 20, - >$.

Um espaço de tuplos é uma estrutura de memória partilhada associativa[5], a que todos os processos têm acesso independentemente da sua localização. Neste espaço não existe necessidade de estabelecimento de canais directos de comunicação entre os intervenientes, sendo a comunicação efectuada através de escrita e leitura de tuplos do espaço. Como já foi referido, e ao contrário do que se passa na maioria dos sistemas de comunicação por mensagens, nos quais a mensagem deve ser dirigida explicitamente a um destinatário, no modelo Linda o emissor não necessita de explicitamente nomear o receptor (ou receptores) da informação. Ou seja, a informação, sob a forma de tuplo, pode ser lida por qualquer entidade que tenha acesso ao espaço. Existe, neste modelo, um completo isolamento entre o emissor e o receptor (ou receptores) de mensagens. A capacidade de coordenação que caracteriza o modelo assenta na atomicidade das operações de acesso ao espaço, segundo a qual um tuplo é removido do espaço, actualizado localmente pela entidade que invocou a remoção e finalmente é reposto, no espaço, com o novo valor [WCC04].

O modelo Linda inicialmente proposto, era bastante simples ao nível da expressividade dos tuplos e das primitivas de acesso ao espaço, existindo somente três operações (atómicas) de manipulação de tuplos [Gel85]:

- **out**: um produtor emite um tuplo, para o espaço de tuplos, ou seja insere um novo tuplo no espaço de forma assíncrona, ficando este acessível;
- **in** e **rd**: os consumidores podem, através da especificação de um "template" de tuplo, ir buscar dados ao espaço, através das operações **in** (lê e remove tuplo do espaço) ou **rd** (lê tuplo).

O "template" utilizado como argumento para a operação de leitura tem que ser compatível ("matched") ao do tuplo ou seja, os *tipos* dos campos e o seu número têm que ser iguais, embora possam não se encontrar completamentamente instanciados. Ou seja, na leitura (com ou sem remoção) é devolvido um tuplo do espaço compatível com o tuplo de pesquisa. Quando da leitura, e na situação em que existe mais do que um tuplo que satisfaça as condições, não é especificado qual o tuplo retornado[6], sendo somente garantido que é compatível com o "template". Da mesma forma, se mais de

[5] A natureza associativa deste espaço deve-se ao facto de os tuplos serem pesquisados com base nos seus valores e estrutura e não no seu endereço.

[6] O tuplo retornado é seleccionado de forma não determinística.

um processo consumidor efectuar a chamada da operação **in**, com o mesmo "template" em simultâneo, não é possível determinar qual é o processo que "fica" com o tuplo.

Um exemplo simples de um padrão de comunicação possível através do espaço de tuplos, pode ser observado na figura 3.1, em que uma entidade computacional coloca um tuplo no espaço e outra efectua a sua remoção do espaço.

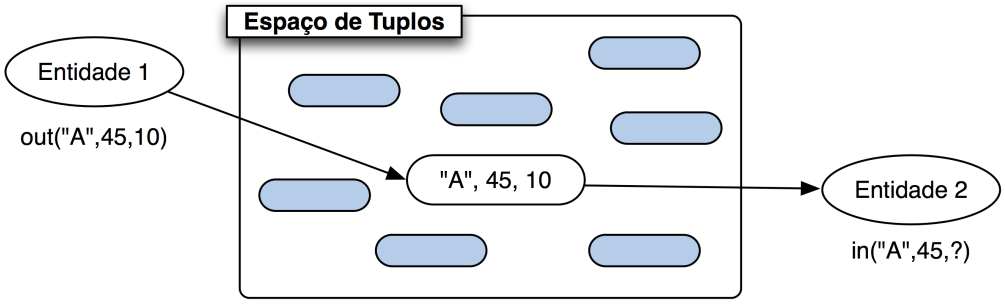

Figura 3.1: Exemplo simples de um padrão de comunicação através do espaço de tuplos

As primitivas de leitura de tuplos, inicialmente propostas possuem uma semântica bloqueante, em que a operação de leitura fica em espera até que exista, disponível no espaço, um tuplo que satisfaça as condições. Pode, desta forma, forçar-se uma sincronização entre o produtor e consumidor de informação.

Posteriormente, foram definidas primitivas de leitura de carácter não bloqueante (**inp** e **rdp**) [Car87]. Estas primitivas, em vez de bloquearem se não for detectado nenhum tuplo que satisfaça as condições, retornam, de imediato, um valor com a indicação de que nenhum tuplo foi encontrado [MW98].

Resumo de características

Uma das principais características do modelo Linda é o facto das comunicações serem totalmente anónimas, ou seja, uma entidade que acede ao espaço partilhado para ler ou remover um tuplo, não tem necessidade de designar explicitamente o processo emissor do tuplo, e vice-versa.

Os tuplos presentes no espaço partilhado têm uma vida independente das entidades que os lá colocaram. Isto significa que, quando uma entidade coloca um tuplo no espaço, através de uma operação de escrita (**out**), este ficará no espaço partilhado até que alguém o remova, independentemente de a entidade responsável pela sua produção ainda existir ou não.

3.3.2 Algumas propostas de sistemas baseados em espaços de tuplos

A abstracção do espaço de tuplos, com a sua interface de alto nível, simplifica bastante a coordenação entre processos distribuídos, sendo uma das suas vantagens a indepen-

dência relativamente à linguagem de programação utilizada pelos clientes.

Diversos sistemas que suportam a implementação de espaço de tuplos têm vindo a ser desenvolvidos [Wel04], sendo alguns dos exemplos, os seguintes:

- JavaSpaces: implementação em Java desenvolvida pela Sun, incorporada na plataforma Jini [Edw01];
- TSpaces: sistema desenvolvido pela IBM [TSp];
- GigaSpaces: uma implementação comercial do sistema JavaSpaces, ao qual foi acrescentado um conjunto de funcionalidades e operações que lhe permitem obter uma maior expressividade na utilização dos tuplos [gt06];
- Klava: sistema de nível *middleware*, especialmente vocacionado para o desenvolvimento de aplicações distribuídas com mobilidade. Possui a noção de localidade (*locality*) que permite identificar de forma única um nó na rede. O sistema implementa o espaço de tuplos com base numa semântica cliente-servidor, em que cada nó pode ser servidor de um espaço de tuplos e, ao mesmo tempo, ser cliente de outros nós, ou espaços de tuplos [BNP02];
- ObjectSpace: implementado em C++, incorporou a noção de objecto ao modelo Linda standard. Nesta implementação qualquer objecto C++ pode ser um tuplo. O esquema de pesquisa de tuplos, neste caso, considera que encontrou um tuplo que satisfaça as condições, se o objecto do tuplo puder implementar o objecto do "template" que é passado para pesquisa.

O modelo JavaSpaces [FHA99, Jav06] assume uma aplicação distribuída como uma colecção de processos que cooperam entre si pela troca de objectos através de um ou mais espaços de partilha; possui mecanismos que garantem a integridade das transacções e permitem lidar com falhas; oferece uma interface de programação simples através da qual é possível manipular os objectos nos espaços, sendo as suas funcionalidades baseadas nas do modelo Linda:

- $write()$: escreve um novo objecto no espaço (semelhante a **out** em Linda);
- $take()$: remove um objecto do espaço (semelhante a **in** em Linda);
- *read*: efectua um cópia de objecto num espaço (semelhante a **rd** em Linda);
- $notify$: notifica um objecto específico, quando uma ou mais entradas que satisfaçam um dado "template" são escritas no espaço (implementações posteriores de Linda, também possuem uma operação semelhante).

Os objectos são passivos, ou seja não podem ser modificados directamente pelos processos, nem os seus métodos podem ser invocados directamente. Para alterar um

objecto, um processo deve proceder à sua leitura do espaço, modificá-lo e tornar a inseri-lo no espaço.

A plataforma TSpaces [WMLF98] desenvolvida sobre Java, constitui uma extensão considerável ao modelo Linda original, conjugando a existência de um espaço de partilha com outros mecanismos de interacção. Todas as interacções entre produtores e consumidores são mediadas pelo espaço de tuplos, possuindo um maior número de operações de escrita e leitura de múltiplos tuplos, do que o modelo Linda. Permite-se a sincronização e troca de dados directa entre dois processos, através de um novo operador (`rhonda`). Possui um mecanismo de eventos que lhe permite enviar uma notificação quando um dado tuplo é escrito ou lido/removido do espaço.

3.4 Modelos de comunicação baseados em eventos

Ao longo desta secção é feita uma breve descrição dos modelos de comunicação em sistemas distribuídos baseados em eventos.

O modelo de comunicação baseado em eventos corresponde a um paradigma de interacção assíncrona entre elementos, em ambientes distribuídos heterogéneos, em que um evento representa uma transição de estado que ocorre e em consequência da qual é emitida uma sinalização por parte de uma entidade a um conjunto de outras entidades [TR05].

Embora este modelo de comunicação já tenha surgido há diversos anos, tem vindo a ser "redescoberto" como uma forma de interacção entre elementos distribuídos dinâmicos e móveis, devido as suas características e capacidade de lidar com o carácter não deterministico e altamente dinâmico das comunicações existentes nestes ambientes [MC05]. Os modelos de comunicação baseados em eventos pretendem resolver algumas das limitações do paradigma cliente/servidor tradicional, onde as interacções eram de natureza essencialmente síncrona e essencialmente direccionadas para comunicações de um para um.

A comunicação baseada em eventos revela-se especialmente útil em aplicações distribuídas que necessitam de reagir a alterações que ocorram em qualquer componente, e de notificar desse facto os restantes componentes. Trata-se de um modelo de comunicação, especialmente bem adaptado para servir de suporte à cooperação entre comunidades de entidades distribuídas que pretendem estabelecer comunicações de forma dinâmica e não previsível ao longo do tempo [MC05].

Os sistemas de eventos têm na sua base um conjunto de elementos produtores de eventos e um conjunto de receptores de eventos e podem ser decompostos nas seguintes fases:

1. ocorrência de evento: trata-se da ocorrência do evento propriamente dito, sendo

resultado de uma alteração de estado;

2. detecção de evento: detecção por parte dos "sensores", do sistema da ocorrência do evento;

3. transmissão de evento: transmissão para os receptores, da informação referente à ocorrência do evento;

4. recepção de evento: tratamento do lado receptor, do evento que foi transmitido.

Nestes sistemas, um evento é uma representação de uma transição discreta de estado que ocorreu e que é sinalizada por uma entidade do sistema, para um conjunto de outras entidades [TR05]. A forma como os eventos são transmitidos depende das opções tomadas na definição do sistema e pode ter por base diferentes paradigmas de comunicação tais como: troca de mensagens, invocações remotas e memória partilhada distribuída. A troca de mensagens representa uma descrição de mais baixo nível em que o modelo de comunicação recorre ao simples envio e recepção de mensagens entre os intervenientes. No caso da invocação remota, a comunicação é efectuada através da chamada, por parte do emissor, de um procedimento ou método declarado nos receptores [EFGK03]. Na memória partilhada distribuída, os vários produtores podem colocar informação, de forma assíncrona, no espaço comum, onde os consumidores a podem ir buscar de forma síncrona [Gel85].

No entanto, uma das formas mais interessantes de comunicação em sistemas de eventos, devido às características que exibem, tem por base o paradigma de publicação/subscrição.

3.4.1 Publicação/subscrição

A comunicação através do modelo publicação/subscrição suporta a interacção de n para n entidades, onde os produtores não necessitam de endereçar directamente os consumidores e vice-versa [Muh02]. Os possíveis consumidores indicam (subscrevem) a informação em que estão interessados, sendo da responsabilidade do sistema encaminhar essa informação, dos produtores aos consumidores (figura 3.2) [Pie04].

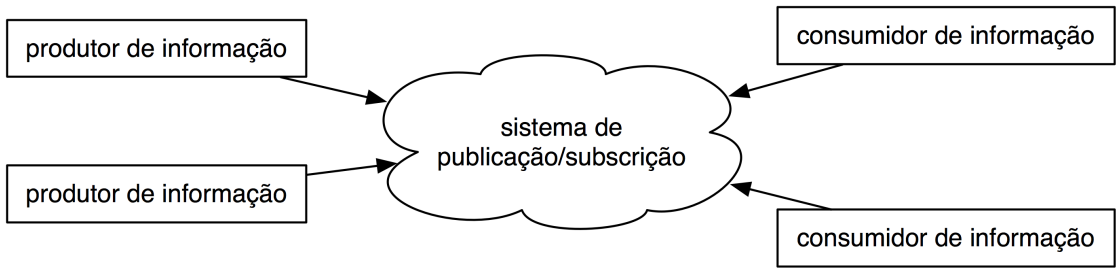

Figura 3.2: Publicação/subscrição

No paradigma publicação/subscrição não é necessária a interacção directa entre as entidades produtoras e consumidoras, existindo um desacoplamento entre estas ao nível do tempo, espaço e sincronização [EFGK03].

O modelo de eventos é, como já foi referido, constituído por dois tipos de entidades, para além do encaminhador, usualmente denominadas de produtores e consumidores (receptores de eventos). Seguindo as interacções entre elas (ver figura 3.3), de um modo geral, tem-se o seguinte padrão de comportamento:

- um consumidor demonstra o seu interesse num tipo de evento, suportado por uma determinada entidade produtora, ou seja subscreve aquele tipo de evento;
- ao detectar as condições que desencadeiam um determinado evento, a entidade produtora gera (publica) o evento;
- como consequência desta publicação, é desencadeada a sequência de notificação dos consumidores, que subscreveram aquele tipo de evento.

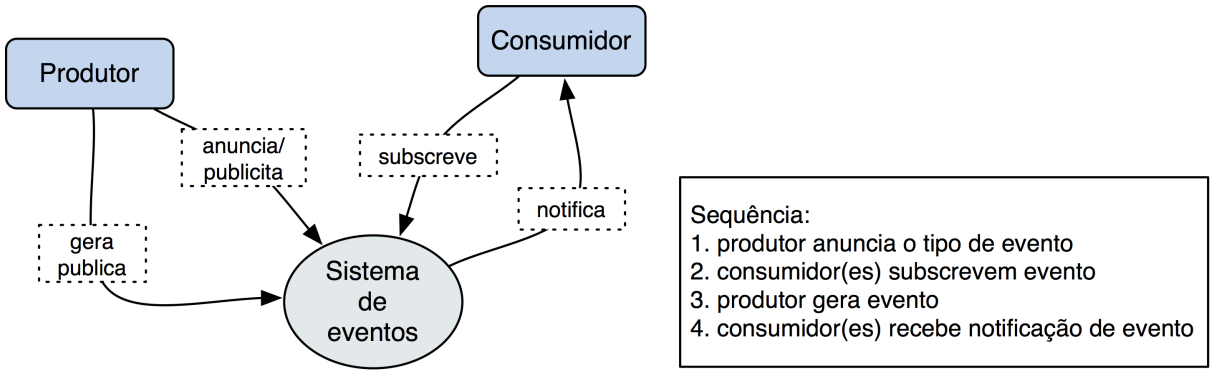

Figura 3.3: Sistema de eventos

Com base neste modelo de eventos, é estabelecida a forma como os produtores e consumidores podem interagir e que pode ser resumida do seguinte modo:

- produtor:
  - criar e definir tipos de eventos;
  - desencadear eventos;
  - distribuir eventos aos interessados;
- consumidor:
  - subscrever tipos de eventos;
  - reagir a eventos, quando estes são recebidos;
  - remover a subscrição de eventos, anteriormente subscritos.

3.4.2 Notificação

A notificação é a forma como é entregue, por parte do sistema, o "evento"aos consumidores. A notificação contém informação referente ao evento, incluindo informação relativa às circunstâncias em que este ocorreu. A estrutura que representa a notificação traduz a expressividade de um sistema de publicação/subscrição, sendo um factor condicionante no tráfego de comunicação envolvida.

O processo de notificação dos consumidores pode ser gerido:

- pelo serviço de notificação, que pode manter uma lista actualizada dos clientes que subscreveram o evento, sendo a notificação emitida somente para os que satisfaçam as condições da subscrição;
- pelos consumidores, ou seja a selecção de quais as notificações vai processar fica da responsabilidade do consumidor, através por exemplo de um filtro.

Na situação em que a selecção das notificações é gerida pelos clientes consumidores existe um maior tráfego de comunicações envolvido embora a estratégia de entrega seja muito simplificada (notificações são entregues a todos os participantes). Na situação em que as notificações são geridas pelo serviço de notificações é exigida uma estratégia de entrega mais complexa, mas menos tráfego de comunicações envolvido.

3.4.3 Algumas propostas de sistemas de eventos

Em termos de implementação, os sistemas baseados em eventos podem, de um modo geral, ser encarados como uma extensão das plataformas intermédias suportadas por troca de mensagens [CNF98], tal que a distribuição de mensagens é efectuada com base em filas explicitamente definidas que garantem a entrega das mesmas.

Como já foi referido, tem havido intensa investigação sobre os modelos de disseminação de eventos e sobre as suas implementações. Como exemplos de alguns, de entre muitos, sistemas de eventos nomeiam-se os seguintes: CEA [BMB+00], JEDI [CNF01], Siena [CRW01], Hermes [PB02, Pie04]. Diversos destes sistemas, constituem extensões a plataformas já existentes como por exemplo o CORBA ou Java RMI.

No caso do sistema CEA - *Cambridge Event Architecture*, os produtores disponibilizam (publicam) a interface que contém os eventos que cada um tem capacidade de notificar, como se pode observar na figura 3.4. O objecto possui um método de registo na sua interface, que possui parâmetros para especificar o tipo de evento e caracteres genéricos (*wildcards*). A ocorrência de um evento é traduzida na criação de um objecto de um tipo específico. Eventos mais complexos são tratados por mediadores de eventos que actuam como intermediários entre os produtores e os consumidores de eventos e que possuem a capacidade de detectar este tipo de eventos.

O sistema JEDI - *Java Event-based Distributed Infrastructure*, tem uma arquitectura distribuída que é constituída por um conjunto de servidores distribuídos (*dispatching*

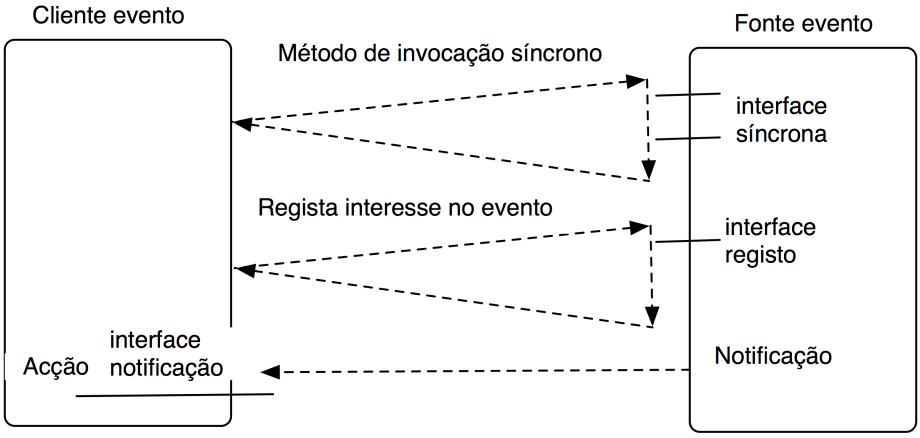

Figura 3.4: Arquitectura de eventos publicação-registo-notificação - CEA [BMB+00].

servers) que estão ligado numa estrutura em árvore. Esta infra-estrutura tem por base a noção de Objecto Activo (OA), que se trata de uma unidade computacional autónoma que desempenha uma determinada tarefa específica da aplicação. Cada objecto activo possui o seu fluxo de execução (*thread*) próprio e interage com os restantes OAs através da produção e consumo de eventos. Esta arquitectura tem vindo a ser expandida de modo a suportar clientes (produtores ou consumidores), com mobilidade e configurações de redes *ad-hoc*.

O sistema Siena - *Scalable Internet Notification Service*, foi desenvolvido como um serviço de notificação para a Internet. Ao mecanismo de publicação/subscrição base foi acrescentada a funcionalidade de anúncio que é utilizada como forma dos produtores de eventos anunciarem quais as notificações de eventos têm capacidade de publicar (figura 3.5).

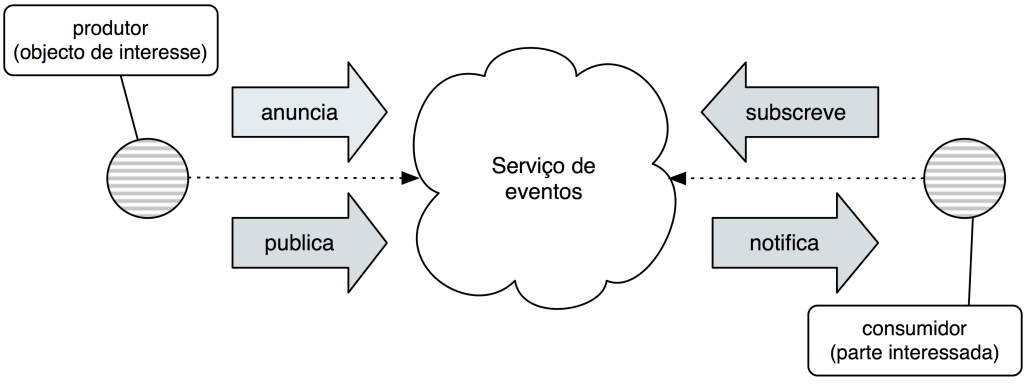

Figura 3.5: Serviço de notificação de eventos distribuída, Siena [CRW01].

O sistema Hermes [PB02] é um sistema de eventos com interacção ponto-a-ponto, tendo também por base um modelo com semântica de anúncio, tal como o sistema anterior (Siena). Utiliza pontos de sincronização (*rendez-vous* - RP), como forma de coordenar a propagação de anúncios e de subscrição [TR05].

O sistema Jini [Edw01] trata-se de uma plataforma que permite aos programadores de aplicações distribuídas, desenvolverem serviços cuja descrição é anunciada e descoberta pelos intervenientes no sistema, sendo possível a comunicação de eventos entre processos distribuídos, suportados por JVM (Java Virtual Machines) distintas. O sistema Jini possui uma interface que lhe permite receber chamadas remotas de instâncias de uma classe de eventos. Um objecto do tipo evento remoto contém a referência ao objecto Java onde ocorreu o evento e um identificador que identifica o tipo de evento. Uma classe do tipo gerador de eventos remotos (RemoteEventNotification) aceita registos de objectos e retorna uma instância da classe que suporta o registo de eventos (EventRegistration), para gerir os registos. Seguidamente envia um objecto de evento remoto (RemoteEvent) para todos os interessados.

3.5 Conclusão

Neste capítulo foram analisados diferentes modelos de interacção, com diferentes características e oferecendo distintas funcionalidades, que se podem revelar importantes para a modelação e desenho de aplicações como as referidas no capítulo anterior.

No caso dos modelos de grupos, as propostas e plataformas existentes, oferecem já importantes garantias quanto às semânticas de ordenação na entrega de mensagens, à gestão da consistência das visões entre os membros do grupo e à consideração de situações de ocorrência de falhas. De uma forma global, os modelos de grupos oferecem um enorme potencial, se considerados a um nível de abstracção superior ao disponibilizado por aquelas plataformas de nível intermédio. Consideramos que esse potencial está ainda, em grande parte, por explorar, sobretudo se considerarmos os contextos de aplicações emergentes, cada vez mais mais frequentes nos nossos dias, em que múltiplas entidades a distribuídas estabelecem, de forma dinâmica, interacções entre si, baseadas em distintos paradigmas de comunicação.

O modelo que se desenvolveu e explorou neste trabalho, tira partido, por um lado, do conceito de grupo, como unidade de estruturação/organização das entidades que constituem uma aplicação e por outro lado, permite confinar, a uma colecção de entidades cooperantes que visam partilhar informação comum, o alcance das interacções que estabelecem entre si. Ao associar o conceito de grupo, a um espaço de interacção caracterizado por entidades que partilham "interesses" ou "objectivos" comuns, estamos não só a contribuir para uma mais fácil compreensão da dinâmica de aplicações complexas, como estamos a facilitar o seu desenvolvimento, de forma estruturada, modular e incremental. Uma discussão mais detalhada destas dimensões é apresentada no capítulo seguinte.

No caso dos espaços partilhados, a evolução verificada, em particular desde a pro-

posta do modelo Linda, foi no sentido de, progressivamente, reconhecer a importância de um mecanismo que suporta a comunicação anónima indirecta entre entidades, facilitando as suas interacções no contexto dos sistemas distribuídos assíncronos. Igualmente, as características desses modelos, no que respeita à coordenação de processos distribuídos, derivadas da natureza global do espaço de tuplos e atomicidade e semânticas das operações básicas *out*, *in* e *rd*, tornam esta abordagem mais interessante, a ponto de se ver a sua influência e integração numa diversidade de ferramentas e plataformas de nível intermédio, bem como a nível de modelos de coordenação e em linguagens de programação de mais alto nível.

A principal dificuldade desta abordagem, a sua realização em ambientes de execução distribuída, veio a ser enfrentada em múltiplos trabalhos nos últimos anos, explorando, por um lado, diversas estratégias de representação (por exemplo, repartição e ou replicação) dos tuplos e diferentes estratégias de implementação das operações *out*/*in*, face às acções de actualização dos tuplos e envolvendo, por um lado, tentativas de adequar essas estratégias, aos padrões mais frequentes, das operações de acesso ao espaço, por parte das aplicações.

A natureza não estruturada do espaço de tuplos, na sua proposta original, constitui uma dificuldade adicional, se se pretender adaptar este paradigma a sistemas de grande escala, ou que apresentem uma diversidade considerável de entidades que evoluam dinamicamente e interajam entre si. Algumas propostas foram feitas, no sentido de, por exemplo, permitir uma aplicação ser organizada em termos de múltiplos espaços de tuplos, cada um identificado por um nome global único.

No trabalho que se desenvolveu nesta dissertação, explorou-se uma abordagem que consideramos mais expressiva e que consiste em associar a cada grupo, entendido como uma entidade de estruturação da aplicação, um espaço de tuplos próprio apenas acessível aos membros do grupo. Esta associação é compatível com a noção do grupo como um espaço confinado de interacção, onde, para além da difusão, encontramos agora também o acesso ao espaço partilhado, comum aos membros do grupo. A exploração deste conceito, ao nível das aplicações como as identificadas no capítulo 2, é uma das contribuições do trabalho aqui apresentado.

Naturalmente, as formas de interacção assíncronas, tais como as baseadas na notificação por eventos, são enquadráveis também no controlo das comunicações confinadas a cada grupo.

A combinação das dimensões acima mencionadas e a sua integração num modelo (MAGO), suportado por uma arquitectura assente numa plataforma de computação distribuída, são discutidas com detalhe nos capítulos seguintes.

4

Um modelo de computação baseado em grupos

Conteúdo

Neste capítulo são discutidos e definidos os conceitos e primitivas associados ao modelo. Numa primeira fase são identificados os objectivos e pressupostos do modelo proposto e de seguida são analisados os seus elementos constituintes. São também discutidos os mecanismos de interacção definidos assim como a sua utilização através de breves exemplos das primitivas do modelo, em pseudo-código. No final do capítulo, são discutidos dois cenários possíveis de utilização do modelo, que ilustram a forma de modelar as interacções entre os utilizadores presentes em diferentes ambientes.

4.1 Introdução

O objectivo principal do modelo proposto (MAGO - "Modeling Applications with a Group Oriented approach") é facilitar o desenvolvimento de aplicações que lidam com grupos de entidades móveis que interagem, partilham dados e comunicam entre si dinamicamente. O domínio considerado para um cenário típico de utilização envolve aplicações em que múltiplas entidades móveis interagem entre si, agregando-se dinamicamente em grupos, que tendem a explorar características específicas de comunicação e cooperação, assim como as particularidades de interacção com o meio envolvente. O foco do trabalho desenvolvido situa-se na modelação da organização e interacção estabelecida entre diversos elementos com recurso aos paradigmas de grupos.

O modelo aqui proposto segue a linha de investigação iniciada com o modelo GroupLog [BC01], onde foi apresentada uma especificação, numa linguagem de programação em lógica que se enquadrou no contexto de um primeiro exercício de validação do modelo. O modelo GroupLog é um modelo baseado em grupos, que permite a organização de um sistema em grupos de entidades [Bar04], possuindo mecanismos e abstracções que suportam a especificação da coordenação das entidades. Um sistema em GroupLog é composto por um número variável de entidades concorrentes, as quais podem formar grupos de modo a participarem em actividades de coordenação. Os grupos (denominados neste modelo como entidades compostas) permitem modelar unidades de cooperação e também estruturar o espaço de entidades no que diz respeito à sua organização e por conseguinte, às dependências existentes entre as entidades. No GroupLog os grupos, bem como as entidades elementares, são caracterizadas por um nome, uma interface bem definida e um programa. Os grupos possuem um espaço partilhado sendo este representado por um conjunto de tuplos. A interacção entre entidades, dentro de um grupo, é suportada por duas formas básicas de comunicação (figura 4.1):

- acesso ao espaço partilhado do grupo, através de funções pre-definidas, baseadas no modelo Linda;
- invocação de métodos da interface das instâncias que sejam membros do grupo, ou seja por comunicação directa entre entidades.

A interacção entre uma entidade do exterior e entidades elementares pertencentes a um grupo, é efectuada sempre através da interface do grupo.

No desenvolvimento do modelo MAGO, retiveram-se alguns dos aspectos essenciais do modelo GroupLog, em particular relativamente à definição de grupos e à partilha de um espaço de tuplos associado a cada grupo, como uma das formas de interacção internas ao grupo. No entanto, face aos estudos efectuados sobre os perfis das aplicações, envolvendo interacções baseadas em dispositivos móveis que representam

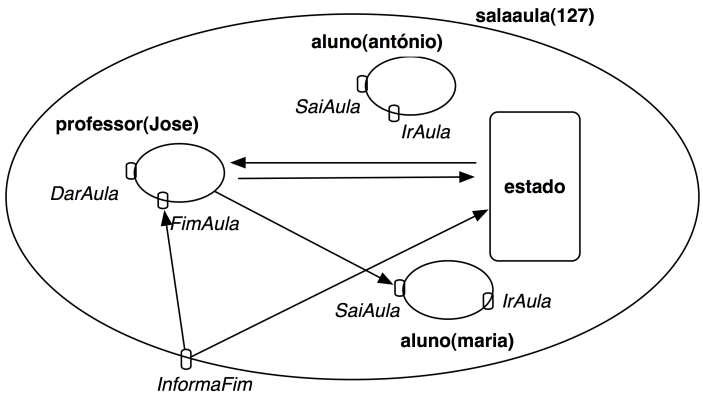

Figura 4.1: Interacções dentro da entidade composta [Bar04]

utilizadores que estabelecem diversas formas de cooperação e partilha de informação, foram considerados aspectos, que permitam dar uma maior expressividade ao modelo de grupos. Esta expressividade teve como objectivo suportar o desenvolvimento de aplicações sobre ambientes de computação móvel, em que coexistem diversos tipos de utilizadores que pretendem de alguma forma organizar-se entre si de acordo com as suas características e preferências (grupos implícitos).

Os grupos de utilizadores podem ser definidos e criados de forma dinâmica, permitindo as entradas e saídas de membros em qualquer momento de vida de um grupo. Esta é uma necessidade que está directamente relacionada com o tipo de utilização, em ambientes onde os utilizadores se movimentam livremente, podendo em qualquer instante optar por sair ou entrar em grupos (ou mesmo formar novos grupos).

Foram disponibilizados, neste modelo, novos mecanismos de comunicação para a interacção entre entidades elementares e grupos, em particular os que permitem a difusão de informação para um grupo de entidades.

O modelo aqui proposto define um conjunto de primitivas e funcionalidades que facilitam a concepção de aplicações, tirando partido da organização em grupos. Um dos objectivos foi o de permitir, a quem desenvolve aplicações, abstrair-se dos problemas relacionados com a gestão interna dos grupos e ter acesso a distintas formas de comunicação.

No modelo MAGO são apresentadas novas propostas de conceitos e formas de interacção entre entidades, relativamente ao modelo GroupLog, que permitem modelar aplicações interactivas distribuídas de um modo simples:

- comunicação directa entre entidades através de mensagens e da invocação de métodos definidos na sua interface;
- comunicação de um para muitos onde a difusão de informação dentro de um grupo é efectuada através de mecanismo de eventos;
- utilização do espaço partilhado como forma de oferecer um espaço de partilha

de informação a todos os membros de um grupo;

- definição do conceito de grupos implícitos que, com base no mecanismo de eventos e do acesso aos dados próprios de cada entidade elementar, permite a criação automática de grupos de acordo com um conjunto de preferências definidas ao nível da aplicação;

- definição de um modelo que se enquadra num contexto orientado a objectos (GroupLog foi desenvolvido num contexto de programação em lógica).

O modelo MAGO continua a oferecer garantias de consistência[1] ao nível das primitivas de comunicação entre as entidades. No entanto, a possibilidade de definição de hierarquias de grupos foi deixada para ser gerida pelas aplicações. Esta opção deveu-se ao facto de se pretender flexibilizar e simplificar a forma como o modelo lida com a estrutura de organização e também porque o modo como as entidades tendem a organizar-se é muito dependente de aplicação para aplicação. Outra opção oferecida pelo modelo é a possibilidade de comunicação directa com entidades, mesmo quando estas são membros de um grupo, ou seja as entidades mantém a sua visibilidade mesmo quando se tornam membros de um grupo, não sendo necessário estabelecer a comunicação através do grupo.

O modelo proposto parte de certos pressupostos, tais como:

- assumir que existe uma infraestrutura de rede que suporta as comunicações e detecta as possíveis falhas de comunicação;

- questões associadas à segurança, ainda que sejam um problema importante, não foram contempladas neste trabalho.

4.2 Aspectos conceptuais

Como já foi referido no capítulo anterior, associados aos modelos de grupos apareceram, nas últimas décadas, diversas plataformas de programação suportando abstracções para gerir a constituição dinâmica de grupos de processos, a consistência das visões observadas pelos membros de um grupo e diversas ordenações de mensagens, preservando a causalidade, atomicidade ou ordenação total.

O trabalho que se desenvolveu nesta dissertação enquadra-se numa direcção de investigação que procura explorar os conceitos dos grupos como paradigmas de organização e cooperação, mas situando-se a um nível de abstracção superior. Diversas plataformas como o Isis, Horus, Ensemble ou mais mais recentemente o JavaGroups e o JGroups, entre outras, disponibilizam já funcionalidades suficientemente genéricas

[1] Esta consistência é garantida pela plataforma de suporte utilizada (JGroupspace) e é mantida pelo modelo MAGO.

a um nível intermédio (*middleware*). Estas plataformas assentam em implementações suficientemente robustas para poderem servir de base à realização da abstracções de grupo que sejam disponibilizadas, quer ao nível de linguagens de programação de alto nível (como o GroupLog [Bar04]), quer ao nível de modelos como o MAGO, que é apresentado nesta dissertação.

Nesta secção, discutem-se, numa perspectiva mais global, as duas principais dimensões subjacentes a esta abordagem: (1) uma metodologia para a especificação de grupos; e (2) identificação e descoberta dinâmica de grupos.

1) Uma metodologia para a especificação de grupos

No sentido de definir um modelo genérico e abstracto, foram seguidos os princípios fundamentais:

- são consideradas duas categorias de entidades, as entidades elementares e as entidades compostas ou grupos, por forma a capturar, de forma separada, os aspectos da computação e os aspectos da cooperação, respectivamente;
- uma entidade elementar representa o aspecto computacional básico, tal como é encapsulado numa entidade autónoma de programação. Deve apresentar uma interface pública bem definida, que esconda o seu comportamento interno. Num modelo genérico, devem ser possíveis diferentes interpretações para uma entidade elementar, dependendo do seu enquadramento a nível do ambiente de programação. Por exemplo, como um processo, um objecto, um agente ou um serviço;
- o modelo computacional interno a uma entidade elementar deve ser "escondido" ao exterior, podendo admitir diferentes interpretações, desde um comportamento reactivo, em resposta à invocações dos pontos de entrada declarados na sua interface, até um comportamento pró-activo, suportado por fluxos de execução autónomos;
- uma entidade elementar composta (grupo), deve igualmente disponibilizar uma interface bem definida, para a interacção com o exterior. Em princípio, do ponto de vista da organização global de um sistema, grupos e entidades elementares deveriam ser, idealmente, semelhantes para um observador externo. A separação da definição da interface de um grupo da especificação do seu comportamento interno, obtido este à custa da cooperação entre os membros de um grupo, permitem realizar estratégias locais ao grupo, de forma transparente aos seus "utilizadores";
- tanto as entidades elementares como os grupos devem admitir múltiplas instâncias, criadas e destruídas dinamicamente, por forma a responder à natureza dinâmica das aplicações interactivas, bem como dos ambientes de computação que

as suportam. No caso dos grupos, a sua constituição deve poder mudar também dinamicamente, à medida que novos membros entram e saem dos grupos. Deste modo, a agregação de entidades em grupos permite explorar formas de cooperação dinamicamente estabelecidas, entre os membros do grupo;

- um grupo encapsula um espaço de interacção confinado aos seus membros correntes. A estruturação de uma aplicação em termos de múltiplos grupos permite, do ponto de vista da arquitectura de um sistema, explorar, por exemplo os benefícios das relações de proximidade entre os seus membros. Por outro lado, a flexibilidade de um modelo de grupos, do ponto de vista do desenvolvimento de aplicações, depende muito da expressividade dos modelos de comunicação que são suportados entre os seus membros. Na generalidade das plataformas de comunicação em grupo de nível intermédio, são suportadas formas de difusão de mensagens internamente ao grupo. No entanto do ponto de vista das aplicações, para além de suportarem a cooperação entre os seus membros, os grupos podem suportar contextos de partilha de informação a qual pode ser consultada/actualizada pelos seus membros. No sentido de suportar as diversas formas de interacção necessárias para o desenvolvimento de aplicações, a abordagem seguida neste trabalho foi a de integrar, no contexto de um modelo de grupos, as seguintes formas de comunicação entre os seus membros: comunicação ponto-a-ponto, por difusão e por acesso a um espaço partilhado interno ao grupo. As formas de comunicação ponto-a-ponto e por difusão adoptam uma semântica de transmissão de mensagens, ou na sua variante de notificação assíncrona, associada ao modelo de eventos publicação/subscrição. A forma de acesso a um espaço partilhado assume uma semântica de espaço de memória, admitindo operações de leitura e escrita, sendo interpretado como um repositório de informação partilhado e global aos membros do grupo, ao qual se podem associar por exemplo, propriedades de persistência, com garantias de acesso consistente, já que é baseado no modelo Linda;

- para garantir que num modelo de grupos, os seus membros possam tomar decisões consistentes, o modelo de sincronia virtual [BJ87], garante que os eventos de modificação da constituição de um grupo e os eventos relativos à entrega de mensagens aos membros do grupo, são apresentados a todos os membros, segundo uma ordem total. Uma vez que cada membro do grupo pode construir localmente uma história da evolução dos estados do grupo através das mensagens que recebe, com informação sobre os eventos que ocorrem no grupo, este modelo de sincronia virtual, permite facilitar a programação de aplicações distribuídas baseadas em grupos. O conceito de sincronia virtual permite que os membros do grupo assumam que os eventos, como os acima mencionados, ainda que ocorrendo de forma concorrente num sistema assíncrono, surjam como virtualmente

sincronizados. Por forma a realizar este conceito, a implementação deve garantir a sincronização entre a instalação de uma nova constituição do grupo e a entrega de mensagens difundidas para o grupo.

No modelo MAGO aqui proposto segue-se, como já foi referido, a abordagem do modelo GroupLog, no que se refere à integração no contexto de um grupo, de formas de comunicação por mensagens e acesso a um espaço partilhado de tuplos. Como consequência, o modelo garante a consistência das visões relativamente aos eventos de entradas e saídas, à difusão de mensagens e ao acesso a espaço partilhado.

2) Identificação e descoberta dinâmica de grupos

Neste trabalho, considerou-se uma segunda dimensão que, do ponto de vista conceptual, explora formas de identificação dos padrões de atributos de entidades num ambiente distribuído, com base na qual se gere de forma automática, a integração das entidades em grupos de interesses.

Em particular, nos cenários de aplicação considerados neste trabalho, tais grupos são definidos através de um registo inicial de uma lista dos seus atributos distintivos, os quais são utilizados por um mecanismo de descoberta, que detecta os potenciais membros e os filia automaticamente nos grupos de interesses de acordo com os seus perfis.

Este conceito, que se designou de grupos *implícitos*, acrescenta expressividade ao modelo MAGO, quando comparado com outros modelos de grupos, na medida em que lhe confere uma capacidade de adaptação dinâmica da configuração dos grupos que integram uma aplicação.

4.3 Entidades do modelo

Nesta secção, são descritas as entidades básicas do modelo. Seguindo os conceitos do GroupLog, o modelo contempla dois tipos de entidades distintas: entidades elementares e entidades compostas (grupos), que irão ser descritas seguidamente.

Qualquer configuração de um sistema[2] baseado neste modelo inclui um grupo universal (GU) que agrega todos os elementos do sistema, de tal forma que qualquer entidade (elementar ou composta) pertence sempre a este grupo (ver figura 4.2). É dentro deste grupo que todas as interacções entre elementos irão ocorrer. Este grupo, tal como os outros que possam vir a ser formados, possui um conjunto de funcionalidades que permitem a interacção entre entidades, a comunicação directa, a difusão de eventos e a partilha de informação.

[2]Neste contexto entende-se por "sistema" MAGO a configuração de uma colecção de entidades elementares e compostas (grupos) com a especificação dos seus comportamentos e interacções, para cumprir os objectivos especificados para suportar uma aplicação. Esta definição corresponde a ter um "sistema" configurado para cada aplicação.

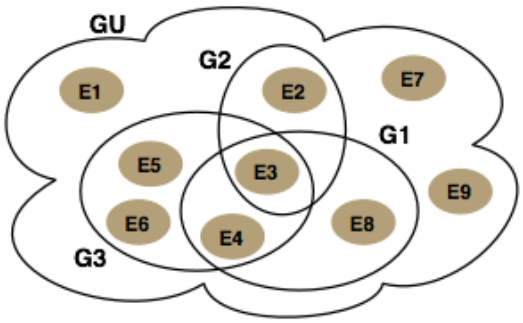

Figura 4.2: Entidades do Modelo

Embora todos os grupos se formem dentro do espaço do GU, o modelo MAGO não possui um suporte explícito que permita gerir hierarquias de grupos. Este aspecto não foi considerado prioritário para o desenvolvimento de aplicações, podendo no entanto ser tratado ao nível da aplicação, já que o modelo proposto não impõe qualquer tipo de condicionalismo a esta estruturação.

4.3.1 Entidades elementares

As entidades elementares são as componentes do sistema que representam entidades computacionais activas, isto é com um comportamento definido por um programa próprio que determina as acções da entidade e as suas interacções com outras entidades. Assume-se que a definição do programa de cada entidade elementar que estabelece o seu comportamento e da interface que a entidade disponibiliza ao exterior, é efectuada no contexto do ambiente de desenvolvimento no qual o modelo MAGO se enquadra. Em particular, no caso deste trabalho, tais definições e instanciações das entidades elementares são feitas no contexto de uma linguagem orientada a objectos (Java).

Cada entidade elementar pode ser caracterizada como um elemento activo, que possui os seguintes atributos (ver figura 4.3):

- *Nome:* um identificador único e global no sistema;
- *Interface:* um conjunto não vazio de métodos, oferecidos pelo modelo e ainda os definidos pelo programador, que permitem a comunicação (interacção) com as restantes entidades componentes do sistema;
- *Atributos próprios:* um conjunto de propriedades, características da entidade elementar;
- *Estado:* o estado das computações desencadeadas pelo programa da entidade. Em particular, o estado da sua memória e da sua interacção com o meio (ou seja, outras entidades presentes no sistema).

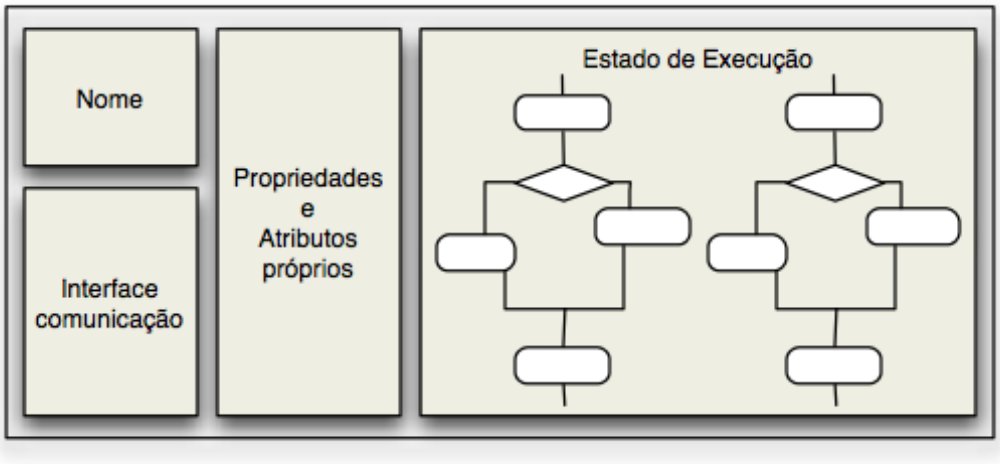

Figura 4.3: Entidade elementar

São seguidamente apresentados, em maior detalhe, os atributos mencionados anteriormente.

Nome Cada entidade presente no sistema possui um identificador, obrigatoriamente único e global, gerido pela implementação do modelo, ao nível da arquitectura que o suporta.

Interface As entidades são independentes entre si e possuem comportamentos próprios. Para a interacção com os restantes elementos do sistema, recorrem aos diferentes mecanismos de comunicação disponibilizados pelo modelo. A interface de comunicação comporta três mecanismos distintos, que podem ser utilizados e combinados de acordo com as diferentes necessidades das aplicações[3]:

- um mecanismo que permite a comunicação directa entre entidades; este mecanismo pode tomar duas formas: a chamada explícita de um método da interface da entidade com a qual pretende estabelecer o contacto; ou a entrega de mensagens, que são posteriormente e explicitamente "tratadas" pela entidade receptora, através de um método definido pelo modelo;

- um mecanismo de comunicação baseado em eventos, disseminados no contexto de cada grupo, que permite à entidade subscrever um serviço de eventos e ser notificada, quando da ocorrência de um evento;

- um mecanismo que permite a partilha de informação entre os membros de um mesmo grupo, através do acesso a um espaço partilhado, interno ao grupo.

[3]Estes mecanismos são abordados em maior detalhe na secção 4.5, referente às forma de comunicação entre entidades.

Ao nível da transmissão, a entidade emissora pode desencadear uma interacção com outra entidade, através da primitiva de envio directo de mensagens ($send()$) ou através de um mecanismo de eventos ($publish()/subscribe()$). A primeira pressupõe uma comunicação directa com o interlocutor enquanto que o mecanismo de eventos permite a comunicação (difusão) para todos os membros de um grupo.

Os diversos mecanismos de comunicação existentes permitem às aplicações adoptarem o tipo de interacção mais adequado às suas necessidades. Múltiplos mecanismos de comunicação podem ser combinados e coexistir ao nível de uma mesma aplicação

Atributos próprios Os atributos de uma entidade permitem definir as sua características próprias. Uma entidade, ao entrar no sistema, define e instancia os seus atributos próprios. Os atributos estão directamente relacionados com a aplicação, dependendo o seu tipo, formato e número, da aplicação que se pretende modelar. O conjunto de atributos de uma entidade é especificado quando da sua criação segundo uma lista de valores, tendo cada instância de cada atributo o seguinte formato:

$$< tipo, atributo, [valores] >$$

Cada campo possui o seguinte significado:

- $tipo$ - categoria do atributo; cada aplicação pode definir um conjunto de categorias, que permitem atribuir uma maior expressividade aos atributos e efectuar uma gestão mais eficiente;
- $atributo$ - nome do atributo, que o caracteriza de forma inequívoca;
- $[valores]$ - lista não vazia de valores que o atributo pode assumir.

Exemplos de definição de atributos, para uma aplicação podem ser:

- $< PESSOAL, NACIONALIDADE, PORTUGUESA >$
- $< PROFISSIONAL, LINGUAS, [PORTUGUES, INGLES] >$

A definição e instanciação dos diferentes tipos de atributos e dos seus valores estão directamente relacionadas com as aplicações, sendo a sua definição, configuração e gestão, da responsabilidade destas.

Estado O estado computacional de uma entidade refere-se ao estado das computações determinadas pelo programa da entidade. Cada entidade prossegue um fluxo de execução que caracteriza a evolução da sua máquina de estados. A transição de estados depende, não só do ponto de execução corrente, como também do estado das suas variáveis (locais e públicas) e do estado das suas comunicações[4].

[4]No estado das comunicações, são consideradas não só as mensagens trocadas, mas também as mensagens que se encontram pendentes para processamento.

O comportamento interno de cada entidade pode resultar de um ou mais fluxos de execução:

- T0: o fluxo principal, que é activado quando da criação da entidade e é o responsável pela execução do programa principal da entidade; este termina quando o programa da entidade acabar a sua execução;
- Ti (com i: 1 ..n): pode ser criado um conjunto de fluxos de execução concorrentes que são activados de acordo com as interacções, sendo cada Ti activado quando um método a ele associado (Mi) for invocado, por exemplo, em resposta a um evento.

O estado computacional de cada fluxo de execução é definido como o conjunto de valores das suas variáveis locais, no seu ponto de execução actual[5]. Cada fluxo de execução passa por uma sucessão de estados lógicos (activo; em espera), conforme se distingue mais à frente (secção 4.4) onde é descrito o comportamento da entidade enquanto membro de um grupo.

Pode então definir-se o estado de computação de uma entidade pelo conjunto dos estados de computação dos seus fluxos de execução (T0, Ti, ...Tn) e pelos valores das suas variáveis globais. Nestas variáveis globais estão também incluídas as estruturas de filas de mensagens pendentes, que caracterizam o estado das interacções entre a entidade e o meio onde esta se encontra.

4.3.2 Entidades compostas - grupos

Numa definição mais genérica, um grupo pode ser visto como "um conjunto de entidades que é considerado como uma unidade" [Bar04].

Para a organização da aplicação, deve ser dada resposta a três importantes questões:

1. como é formado um grupo, ou seja como se constitui/surge;
2. como funciona a entidade grupo, ou seja qual o seu comportamento enquanto entidade (comunicação com o exterior);
3. como se definem as interacções dentro do grupo (comunicação interna).

Assim, enquanto elemento computacional do modelo, um grupo deve dar resposta a essas questões, devendo especificar:

1. mecanismos que permitem a sua criação e a entrada/saída de membros;
2. mecanismos para gerir o estado do grupo, enquanto entidade dependente do estado dos seus membros;

[5]Este ponto, que no fundo define a instrução corrente, é indicado pelo valor dos "registos" internos.

3. mecanismos que permitem a comunicação entre os membros do grupo;

4. Interface de comunicação através da qual o grupo é visto pelo exterior e que permite a interacção entre o grupo e o meio onde este se encontra.

A primeira e a última características especificam a forma como o grupo é visto pelo exterior enquanto entidade, enquanto as restantes definem a forma como este é gerido internamente.

Ao agregarem-se em grupos, as entidades elementares passam a ter acesso às funcionalidades particulares dessa organização, passando a ter acesso a um conjunto de mecanismos que permitem a interacção com os restantes membros do grupo. Uma entidade, ao aderir a um grupo, não perde a sua individualidade, ou seja continua a ser vista como um elemento autónomo, que passa a ter acesso às funcionalidades próprias do(s) grupo(s) a que pertence. No entanto, enquanto membro de um grupo, deve utilizar os mecanismos de interacção deste, devendo especificar qual o grupo a que se refere, dado que cada entidade pode ser membro de vários grupos.

Tendo por base a definição de entidades elementares e as especificações referidas, um grupo é caracterizado pelos seguintes atributos:

- *Nome:* identificador único e global, que permite reconhecer o grupo de forma inequívoca;
- *Interface:* conjunto de métodos definidos que permitem a interacção entre o grupo e o meio;
- *Atributos Próprios:* conjunto de propriedades, características do grupo;
- *Estado:* o estado depende das computações desencadeadas pelos membros do grupo, do estado partilhado interno ao grupo, da sua interacção com o meio e ainda do estado dos seus membros;
- *Conjunto de Membros:* conjunto de entidades que formam o grupo e que se agregam como uma entidade composta.

Nome Cada grupo possui um identificador único e global gerido pela implementação do modelo (tal como as entidades elementares) e também se pode definir um nome simbólico que melhor caracterize e defina o grupo enquanto elemento da aplicação.

Interface O grupo, como entidade do sistema, possui um conjunto de métodos de interface que lhe permite interagir com as restantes entidades e que são associados ao grupo aquando da sua criação. Estes métodos possuem diferentes políticas de atendimento de pedidos e de divulgação para o grupo, que podem ser configurados e/ou redefinidos pelo programador da aplicação. Os mecanismos de comunicação disponíveis são idênticos aos definidos para as entidades elementares, isto do ponto vista

do interlocutor externo, que comunica com o grupo ou com entidades elementares do mesmo modo[6]. No grupo, a gestão das comunicações e de configurações é tratada por uma entidade denominada "Representante do grupo", que tem a seu cargo toda a gestão e tratamento das comunicações entre o grupo e o meio e também dentro do próprio grupo, assim como a gestão dos seus membros.

Atributos Próprios Os atributos de um grupo permitem definir as características e propriedades do grupo. A definição dos atributos de um grupo é idêntica à dada para as entidades elementares e também dependente da aplicação que se pretende modelar. O formato dos atributos de um grupo pode ser configurado de acordo com a aplicação.

Estado O estado de um grupo é definido pela sua constituição interna (ou seja pelos membros que dele fazem parte em cada estado global da computação), pelo estado das suas estruturas de comunicação e pelo estado partilhado do grupo. O estado partilhado do grupo é suportado por uma estrutura de dados global do grupo (baseada no conceito de espaço de tuplos), sendo esta estrutura acessível a todos os membros do grupo.

Conjunto de membros Conjunto de entidades que pertencem ao grupo, ou seja que constituem o grupo. Ao juntar-se a um grupo, uma entidade mantém a sua individualidade e visibilidade, podendo interagir com outros elementos do sistema externos ao grupos e tendo a possibilidade de participar em múltiplos grupos. O facto de uma entidade manter a sua visibilidade, distingue-se do modelo GroupLog no qual uma entidade ao juntar-se a um grupo deixa de ser visível a partir do exterior e só pode ser acedida, de forma indirecta, através da interface do grupo.

Deve salientar-se que, no modelo, certas características do grupo, tais como: **Atributos**, **Estado** e **Conjunto de Membros** podem variar dinamicamente em tempo de execução, o mesmo não se passando com o **Nome** e **Interface**, que se mantêm inalterados após a criação do grupo.

Representante de Grupo

Como já foi referido, os grupos possuem, tal como as entidades elementares, uma interface através da qual interagem com o exterior. A "gestão" dessa interacção é da responsabilidade da entidade "representante de grupo", sendo esta uma entidade que surge devido às necessidades operacionais do modelo, visto que existe a necessidade

[6]Do ponto de vista do exterior, a forma de comunicação com um grupo ou uma entidade elementar é semelhante, sendo da responsabilidade do grupo (internamente) o tratamento da divulgação de informação pelos seus membros.

de definir uma entidade que se responsabiliza por gerir o estado e o comportamento do grupo.

A criação de um novo grupo envolve a "criação" implícita de um representante desse grupo, que passa a "interpretar" o comportamento do grupo. O representante surge como o elemento de ligação entre o exterior e o grupo, centralizando parte das funcionalidades de comunicações e actuando também como elemento gestor de configurações e políticas específicas do grupo.

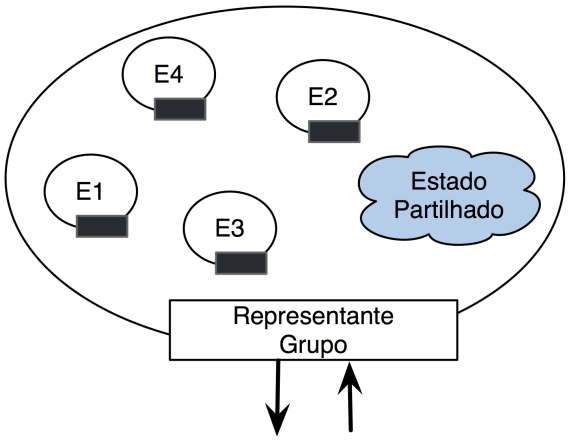

Figura 4.4: Elementos que constituem um grupo

Ao efectuar uma chamada ao grupo, é esta a entidade que é invocada, ou seja um grupo está intimamente ligado ao seu representante, sendo este que permite a comunicação com o grupo como se este se tratasse de uma entidade elementar. Torna-se assim possível, por exemplo, encapsular todo o processamento necessário à difusão das mensagens para os membros do grupo, assim como gerir os mecanismos internos de comunicação. Esta entidade é também responsável pela gestão da admissão de novos membros e pela manutenção da lista de membros filiados.

Em resumo, o representante do grupo é responsável pela gestão dos seguintes aspectos:

- a recepção de mensagens dirigidas ao grupo e a sua difusão pelos seus membros, de acordo com políticas configuradas, quando da criação, para o grupo e conforme a semântica do mecanismo invocado;

- a política de admissão de novos membros (estabelecida quando da criação do grupo) e a promoção dos mecanismos de votação, quando necessário.

Esta entidade é criada automaticamente quando da criação do grupo, que representa e de acordo com as propriedades e características especificadas para este.

O representante de grupo não é uma entidade visível ao nível do programa da aplicação. As suas características determinam os possíveis comportamentos do grupo, na

sua interacção com o exterior, e são implicitamente definidas através dos argumentos passados na invocação da primitiva de criação de um grupo.

Grupos explícitos e implícitos

Em contraste com o modelo GroupLog e com a generalidade de outros modelos anteriores, existem dois tipos básicos de grupos no modelo MAGO: grupos explícitos e grupos implícitos.

Nos grupos explícitos, tanto a sua criação como a entrada/saída de novos membros exigem a invocação explícita, por parte dos candidatos a membros, de primitivas do modelo, sendo a sua filiação validada de acordo com a política de admissão especificada quando da criação do grupo.

Nos grupos implícitos, a sua constituição depende da satisfação de um conjunto de características que devem ser observadas pelos membros registados no sistema. Na definição de um grupo implícito, é inicialmente explicitado o conjunto de características que todos os seus potenciais membros devem possuir. O processo de filiação é depois desencadeado automaticamente e de forma dinâmica pela implementação, para todos os membros que verifiquem o conjunto de características especificado para esse grupo. Nos grupos implícitos, distinguem-se duas fases no processo de criação:

a) definição do conjunto de características que os seus potenciais membros devem possuir; esta constitui a fase de configuração do grupo implícito, ou seja, a definição do seu conjunto de propriedades;

b) momento de filiação dos membros, que pode ser desencadeado em diferentes situações, tais como:

- quando se define e configura um novo grupo, sendo inspeccionadas, de entre as entidades correntemente presentes, quais as que satisfazem as características do grupo;
- quando uma nova entidade se regista no sistema e a sua compatibilidade com as características dos grupos implícitos definidos é verificada;
- quando se efectua uma alteração dos atributos próprios de uma entidade.

Na última situação referida, o facto de alterar os atributos próprios de uma entidade pode desencadear, não só o processo automático de filiação, como o da sua saída de grupos, dado que as condições que validavam a sua filiação nos grupos implícito podem ter sido modificadas.

Como já foi referido, os grupos implícitos possuem características que os distinguem dos grupos explícitos, em particular a forma como é desencadeado o seu processo de criação e a gestão de filiação. Contrariamente aos grupos explícitos, definidos

de forma explicitamente programada pelos seus membros, os grupos implícitos são gerados consoante as configurações projectadas para cada aplicação, sem intervenção directa por parte do programa da entidade (utilizador). A filiação num grupo implícito emerge naturalmente das características individuais do utilizador, que o levam a ser inserido num grupo de entidades com quem eventualmente não tem qualquer tipo prévio de contacto ou que nem mesmo conhece. Utilizações possíveis para os serviços de grupos implícitos podem situar-se por exemplo numa perspectiva de difusão de um possuidor de informação para potenciais interessados, ou seja para um conjunto de elementos que, dado o seu conjunto de atributos próprios, podem ser elegidos como membros de um dado grupo.

A necessidade de existência de um sistema de informação associado à gestão de grupos implícitos, surge pelo facto destes estarem directamente relacionados com os dados próprios de cada entidade, dados estes que são dependentes do tipo de características definidos pelas aplicações. O processo de filiação está intimamente relacionado com a consulta dos atributos próprios[7] das entidades.

Os grupos implícitos são criados ao nível da "administração" do sistema como uma entidade composta (grupo), o que lhes permite comunicar com um conjunto de utilizadores e partilhar com eles informação em distintos suportes audiovisuais, texto ou interagir através de mensagens (tal como para os grupos explícitos).

Um grupo implícito tem um "dono" que, por exemplo, no contexto real de um *campus* ou de um aeroporto, pode ser uma qualquer entidade comercial ou institucional que pretende comunicar directamente com um determinado "tipo" de utilizadores do sistema. O acto de criação de um grupo implícito é explicitamente desencadeado pelo seu "dono", que gera automaticamente um grupo (com as respectivas filiações) das entidades elementares que evoluem no sistema e possuem um determinado conjunto de características que as definem enquanto grupo, não existindo controlo directo na sua filiação, por parte dos membros.

O processo de filiação é desencadeado automaticamente pelas entidades que satisfazem as condições para admissão, especificadas quando da criação do grupo (pelo seu "dono"), sendo posteriormente notificados da sua inserção no grupo. Se uma entidade não pode interferir directamente/explicitamente nos critérios que levaram à sua eventual inclusão num grupo implícito, pode no entanto expressar a sua indisponibilidade para participar nesse grupo, modificando o seu estado enquanto membro do grupo[8].

As principais diferenças, ao nível das funcionalidades, entre os dois tipos de grupos, situam-se no processo de criação e de filiação. Não existe, no caso dos grupos implícitos, uma acção de filiação explícita, sendo a junção de membros ao grupo efectuada durante o acto de criação do grupo. Novas entidades, que surjam posteriormente à criação do grupo e que verifiquem as condições, são também automaticamente filiadas.

[7] Estes atributos próprios são dependentes das necessidades e objectivos das aplicações.

[8] O conceito de estado de entidade enquanto membro de um grupo é explicado na secção seguinte.

As condições definidas para os grupos implícitos estabelecem um conjunto de regras para o grupo[9], o qual deve ser satisfeito por todos os membros admitidos no grupo.

Como se percebe, pela sua própria definição, a gestão de grupos implícitos é em grande parte uma tarefa em que existe uma forte interacção da aplicação com o sistema de informação de suporte, utilizando os dados armazenados sobre as diferentes entidades, para criar de uma forma dinâmica tais grupos.

O conceito de grupo implícito contribui para aumentar o carácter declarativo do modelo MAGO, na medida em que permite desenvolver aplicações interactivas distribuídas sem exigir ao programador que explicitamente invoque as acções de entrada/-saída de grupos por parte das entidades. O conceito apenas exige que o programador caracterize qual o perfil (conjunto de atributos próprios) que as entidades devem possuir para pertencer ao grupo, sendo delegada no sistema subjacente a responsabilidade dessas acções. A realização do conceito pode ser vista como associada a um mecanismo de detecção de eventos relevantes que desencadeia a procura de correspondências entre os perfis das entidades e dos grupos e as acções subsequentes. Deste modo, pode ser capturada a natureza dinâmica e não determinística das aplicações.

4.3.3 Grupo universal

O Grupo Universal (GU) é um grupo, conforme descrito anteriormente, com a particularidade de agregar todas as entidades existentes no sistema. Qualquer entidade elementar ou grupo, que venha a surgir dentro do sistema, pertence sempre a este grupo inicial. Os grupos que venham a ser gerados, são sempre elementos deste grupo universal (GU), que agrega todas as entidades do sistema. Esta estruturação permite uma maior simplicidade na especificação das comunicações entre as entidades do sistema.

Qualquer entidade, ao "entrar" no sistema, passa a ser membro do Grupo Universal, passando a ter acesso a um conjunto de funcionalidades que lhe permitem a interacção com os restantes membros presentes.

A entrada no sistema implica criar uma entidade, à qual é atribuído um identificador único pelo qual a entidade passa a ser universalmente reconhecida. Este identificador é essencial para o estabelecimento de futuras interacções. Para gerir a entrada/-saída de entidades no Grupo Universal existem as primitivas *register*/*unregister*.

Uma entidade, para entrar no sistema, efectua a chamada à primitiva *register*, à qual passa, como argumentos, as suas características e propriedades. Após o registo como membro é atribuído à entidade um identificador (**nome**). A acção de entrada/registo de uma entidade no grupo universal é uma entrada não condicionada, ou seja, não depende de nenhuma autorização por parte dos restantes membros já existentes e activos. Ao efectuar um registo, é indicado o conjunto de atributos que caracterizam a

[9]Expressas como uma conjunção de atributos.

entidade. Esses atributos são definidos ao nível da aplicação, sendo da responsabilidade desta a criação do conjunto de métodos que lhe permitem gerir e manter esses dados ao nível do sistema de informação que tipicamente lhe serve de suporte.

Registo de uma entidade no sistema

register($\underline{in}$: $nome, lista\_atributos; \underline{out}$: $identificador$)[10]

- $identificador$: identificador único atribuído à entidade;
- $nome$: nome simbólico definido ao nível da aplicação;
- [$lista\_atributos$]: lista de atributos, definidos pela aplicação, associados à entidade.

Saída de uma entidade do sistema

unregister($\underline{in}$: $identificador$)

- $identificador$: identificador da entidade a ser removida.

Se uma determinada entidade, registada no sistema, pretende sair definitivamente deste, deve proceder explicitamente à invocação da primitiva *unregister*, à qual deve passar o identificador que lhe foi anteriormente atribuído.

Para as comunicações, o Grupo Universal (GU) tem disponíveis os mesmos mecanismos de comunicação existentes que para os restantes grupos. Ou seja, possui mecanismos de envio directo de mensagens, de espaço partilhado e de eventos. Torna-se assim possível, através de eventos, difundir informação pelos membros registados do sistema. As entidades que se tornam membros do Grupo Universal, podem subscrever e serem notificadas da ocorrência de eventos com origem no espaço do GU.

O facto do Grupo Universal, ser tratado como os restantes grupos permite uniformizar o tratamento efectuado às entidades e grupos, uma vez que todos pertencem a um mesmo grupo.

4.4 Entidade elementar enquanto membro de um grupo

Todas as entidades do sistema interagem enquanto membros de um grupo, como já foi referido anteriormente e todas elas fazem parte de um grupo (o grupo universal).

Nesta secção, numa primeira parte, são descritas as primitivas disponíveis no modelo para a gestão de grupos por parte das entidades, sendo de seguida apresentados os comportamentos e estados possíveis de uma entidade, enquanto membro de um grupo.

[10]Notação $\underline{in}$ e $\underline{out}$, indica parâmetros respectivamente de entrada e de saída.

4.4.1 Primitivas de gestão de grupo

Cada entidade, ao juntar-se ao sistema, tem acesso a um conjunto de primitivas que lhe permitem gerir a sua integração em grupos e a criação de novos grupos. Essas primitivas permitem também gerir o comportamento das entidades dentro de cada um dos grupos a que pertence, partindo sempre a sua invocação de uma entidade. Através das primitivas disponíveis, é possível controlar o modo de operação e estado da entidade dentro de cada um dos grupos a que ela pertence. O modelo define as seguintes primitivas:

- para a criação de eliminação de grupos: $create() \ / \ destroy()$;
- para a entrada e saída de um grupo: $join() \ / \ leave()$;
- para observar a constituição de um grupo: $membership()$;
- para alterar o estado de uma entidade num grupo: $set\_mode()$.

Criação de um grupo

create($\underline{in} : criador\_id, nome\_grupo, tipo\_grupo, tipo\_acesso, max\_membros, [lista\_param]; \underline{out} : identificador$)

Esta primitiva desencadeia o processo de criação do grupo. Após a sua chamada é devolvido um identificador de grupo, que é a identificação atribuída pelo sistema ao novo grupo. A implementação garante que esta é única. Os restantes argumentos definem a configuração e o atributos do grupo, em particular no que se refere à sua política de filiação:

- *identificador*: identificador único atribuído ao grupo pela implementação;
- *criador_id*: identificação da entidade que cria o grupo; esta entidade não passa a pertencer automaticamente ao grupo; para tal a aplicação deverá invocar explicitamente a primitiva de filiação no grupo;
- *nome_grupo*: nome simbólico atribuído ao novo grupo ao nível da aplicação;
- *tipo_grupo*: identifica o tipo de grupo a ser criado (IMPLICIT ou EXPLICIT);
- *tipo_acesso*: identifica a política de filiação que irá ser assumida pelo grupo; existem configuradas as seguintes políticas de acesso[11] de novos membros:

  FREE: entrada sem restrições;

[11] Todas as políticas de acesso existentes pressupõem a existência de um tempo limite de espera ("timeout"); se entretanto não for possível chegar a um resultado, a filiação do candidato a membro não é aceite, sendo retornada essa informação ao cliente.

BY_ONE: qualquer um dos membros do grupo deve validar o pedido de filiação do novo membro;

MAJORITY: a entrada do novo membro deve ser "votada" favoravelmente pela maioria dos membros do grupo;

BY_LIST: a lista de possíveis membros[12], é criada e passada como argumento, na *list_param*, só podendo tornar-se membro quem estiver na lista;

BY_CREATOR: caso particular da validação, em que o acesso deve ser validado pelo criador, que neste caso deve ser membro do grupo;

- *max_membros*: permite configurar o número máximo de membros que o grupo poderá possuir, se for atribuído um valor negativo significa que o grupo não tem limite de participantes;

- *lista_param*: esta lista de parâmetros é opcional e dependente do tipo de grupo. No caso de criação de grupo implícito, este parâmetro é utilizado para a passagem do conjunto de atributos e seus valores que caracterizam a formação do novo grupo, sendo este conjunto utilizado para validar a filiação de novos membros[13]. No caso dos grupos explícitos este parâmetro é vazio (null).

Ao criar um grupo, por invocação da primitiva *create*, é criado um novo grupo com um identificador único atribuído pelo sistema. Este novo grupo, no caso dos grupos explícitos, não possui inicialmente nenhum membro, podendo estes serem posteriormente filiados, por invocação da primitiva *join*. No caso de se tratar de um grupo implícito, após a sua criação, este tem como seus membros todas as entidades elementares que verifiquem o conjunto de valores de atributos, definido para o grupo.

Após a sua criação, um grupo passa a ter definido um representante, que é identificado e associado ao grupo e que é configurado de acordo com os parâmetros passados na invocação da primitiva *create*. O representante de grupo actua como o gestor de grupo, sendo através dele que as interacções com o grupo são efectuadas.

Eliminação de um grupo

destroy($\underline{in}$: $identificador, nome\_grupo, aviso, [mensagem]$)

Esta primitiva "destrói" um grupo existente, removendo o seu registo das estruturas do sistema e eliminando o grupo enquanto entidade computacional, o que se traduz na eliminação do representante de grupo. Somente membros do grupo podem desencadear a sua eliminação. Os argumentos permitem definir qual o esquema de notificação de encerramento que se pretende adoptar:

[12]Os membros são identificados pelo identificador que lhes foi atribuído quando do registo no sistema.

[13]Nos grupos implícitos todos os membros devem possuir o mesmo conjunto de valores de atributos, conforme foi definido na criação do grupo.

- *identificador*: identificação do grupo a remover;
- *nome_grupo*: nome simbólico que foi atribuído ao grupo ao nível da aplicação;
- *aviso*: indica o tipo de notificação que os membros do grupo devem receber, quando da eliminação do grupo, tendo diversas possibilidades:

  NO_NOTIFICATION: não é enviado qualquer tipo de informação explícita aos membros indicando que o grupo irá ser eliminado, sendo da responsabilidade destes lidar com o facto de surgir um erro, ao tentarem aceder ao grupo, ou as possíveis operações em curso sobre o grupo falharem;

  NOTIFICATION: é desencadeado um aviso de eliminação de grupo ao seus membros, sendo da responsabilidade destes lidar com essa informação;

  NOTIFICATION_ACK: é feita uma notificação da destruição do grupo aos seus membros, ficando-se a aguardar que estes saiam do grupo antes de se proceder à eliminação deste;

- *mensagem*: mensagem a enviar para os membros do grupo no caso de encerramento do grupo do tipo NOTIFICATION.

De salientar que no caso de aviso ser do tipo NOTIFICATION ou NO_NOTIFICATION a eliminação do grupo é imediata, sendo as operações em curso sobre o grupo canceladas. No caso NOTIFICATION_ACK o grupo só é eliminado após todos os membros terem saído ou seja o grupo estar vazio[14], não sendo possível a entrada de mais membros no grupo após a operação de *destroy*.

Na situação de eliminação sem notificação (NO_NOTIFICATION) os membros do grupo não recebem informação directa sobre a eliminação do grupo. Os membros dos grupos eliminados deste modo, só detectam que ocorreu uma eliminação quando, ao efectuarem operações sobre o grupo, estas falharem.

As notificações de encerramento de um grupo desencadeiam um evento que é "sentido" por todos os seus membros activos. Na situação em que o membro se encontrava temporariamente "desligado", ao tornar-se activo, este procede à actualização dos seus grupos, removendo da sua estrutura a informação referente aos grupos que entretanto tenham sido eliminados, sendo este processo desencadeado de forma automática.

Entrada de uma entidade num grupo

join($\underline{in} : entidade\_id, grupo\_id, info; \underline{out} : entrada$)

Esta primitiva permite a uma entidade do sistema pedir uma filiação explícita num grupo cujo identificador é passado como argumento. A entrada no grupo obedece à

[14] De modo a evitar bloqueios, é atribuído um tempo máximo pre-definido, de espera para a saída por parte dos membros, findo o qual se procede à efectiva eliminação do grupo.

política de acesso configurada quando da criação do grupo; no caso de confirmação positiva, é desencadeado o processo de actualização da constituição do grupo. Os argumentos desta primitiva são:

- *entidade_id*: identificação da entidade que se pretende filiar;
- *grupo_id*: identificação do grupo;
- *info*: informação fornecida pela entidade; este campo pode ser utilizado para passar informação específica da aplicação, sendo da responsabilidade desta o tratamento dessa informação;
- *entrada*: valor retornado, com indicação do resultado da entrada; no caso de entrada com sucesso, é retornada a identificação atribuída à entidade nesse grupo.

Saída de uma entidade de um grupo

leave($\underline{in}$: $entidade\_id, grupo\_id$)

Esta primitiva desencadeia o processo de saída de um membro do grupo indicado, actualizando a constituição do grupo, sendo os argumentos idênticos aos da primitiva *join*:

- *entidade_id*: identificação da entidade que pretende sair (identificação dentro do grupo);
- *grupo_id*: identificação do grupo;

Obtenção da constituição de um grupo

membership($\underline{in}$: $grupo\_id$; $\underline{out}$: $[lista\_membros]$)

A chamada desta primitiva, por parte de um membros do grupo, retorna a lista dos identificadores de todos os membros, assim como o estado de actividade actual de cada um deles no grupo indicado.

- *grupo_id*: identificação do grupo;
- *lista_membros*: lista com os membros do grupo (identificação e estado de actividade[15]), retornado pela primitiva.

Esta primitiva retorna a lista de todos os membros que se encontram filiados no grupo, ou seja que efectuaram com sucesso uma operação de *join* ao grupo. O estado de actividade de cada um dos membros reflecte o facto destes se encontrarem *online* (activo) ou *offline* (desligado).

[15]O estado de actividade das entidades enquanto membros de grupo, é explicado na subsecção 4.4.2.

Modificação do estado de uma entidade num grupo

set_mode($\underline{in}$: $entidade\_id, grupo\_id, modo$)

Esta primitiva possibilita, a uma entidade, alterar o seu estado de actividade e o modo como é vista dentro do grupo indicado. Os membros do grupo podem alterar o seu estado dentro dos grupos a que pertencem, por exemplo podem desligar-se (passando ao estado designado por OFFLINE), indicando assim que o grupo passa a assumir que o membro se encontra "desligado", ou seja não acessível embora se mantenha filiado no grupo. Os argumentos são os seguintes:

- *entidade_id*: identificação da entidade cujo estado se pretende modificar;
- *grupo_id*: identificação do grupo;
- *modo*: estado de actividade para o qual a entidade deve transitar:

  ONLINE: neste modo a entidade está disponível para interacção com os restantes membros;

  OFFLINE: neste modo a entidade está indisponível e não acessível para comunicações no grupo especificado.

4.4.2 Estados associados às entidades elementares

Nesta secção é feita a descrição dos estados da entidade elementar enquanto elemento computacional e também do seu estado enquanto elemento de um grupo.

Na figura 4.5 ilustram-se as entidades suportadas pelo modelo (entidades elementares e grupos) e a sua relação com os elementos que lhes correspondem nos níveis inferiores da arquitectura computacional que suporta o modelo. No caso das entidades elementares, o modelo assume que existe um programa associado a cada entidade, responsável pelas acções que definem o comportamento da entidade. A execução deste programa é suportada por um processo ou fluxo de execução principal, cuja correspondência com os processos (ou *threads*), a nível do sistema de operação (SO) e sistema de suporte à execução, é da responsabilidade da arquitectura que suporta o modelo (cuja descrição é feita no capítulo 5). Dependendo da infraestrutura computacional subjacente e das características de cada aplicação, podem ter-se distintas correspondências entre os processos que representam as entidades elementares do modelo. Nomeadamente, certos processos podem ser suportados por processos do sistema de operação a nível de servidores que suportam a arquitectura do sistema, e outros processos serão directamente suportados pelo SO a nível de dispositivos computacionais móveis tais como laptops e PDAs. Em particular, no caso destes últimos dispositivos, que exibem características de mobilidade e operação autónoma (e.g. suportados por baterias

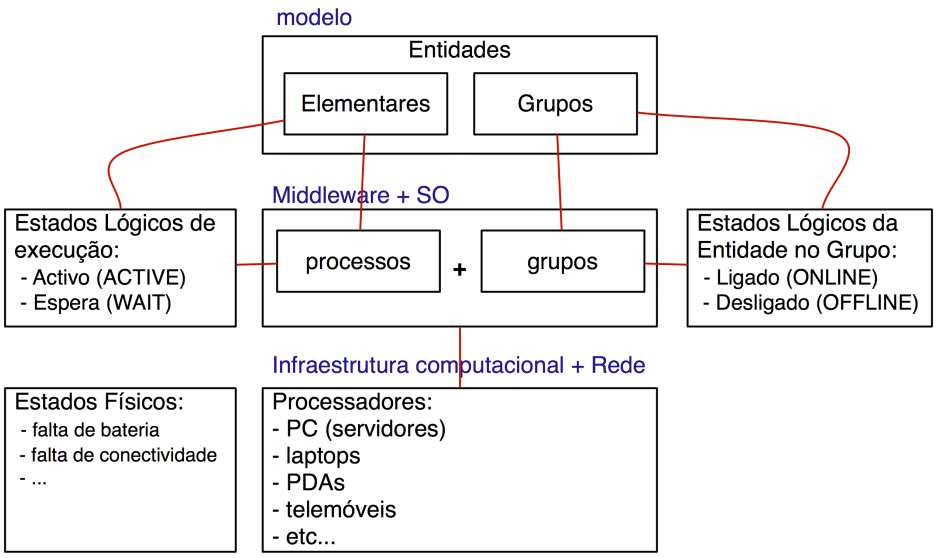

Figura 4.5: Correspondência entre os elementos dos diferentes níveis

locais), o modelo MAGO permite algum controlo ao nível da aplicação, dos estados lógicos das entidades associadas a esses dispositivos.

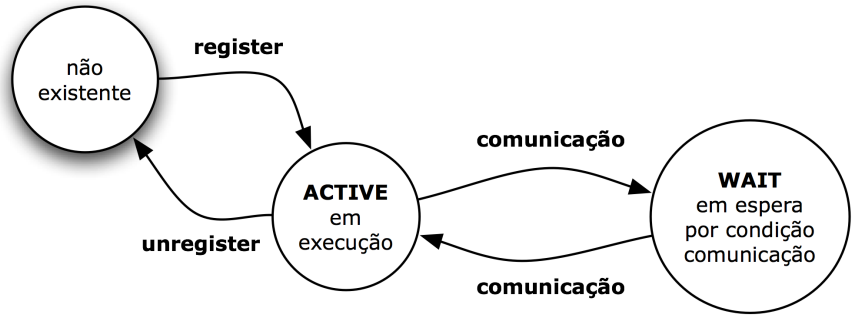

Figura 4.6: Esquema de transição de estados da entidade ao nível da execução.

a) Durante a execução do seu programa, uma entidade elementar passa por diversos estados lógicos, dos quais distinguimos os dois estados principais, que podem ser observados na figura 4.6:

Activo (ACTIVE): Uma entidade, ao entrar no sistema, ou seja ao proceder à acção de registo, passa ao estado de ACTIVE, iniciando a execução o programa que foi previamente definido e que é responsável pelo seu comportamento enquanto elemento do sistema.

Em espera (WAIT): Neste estado, a entidade aguarda pelo completar de uma acção de comunicação com outro elemento do sistema. Enquanto no sistema,

as entidades elementares transitam entre o estado ACTIVE e o de WAIT através de operações de comunicação. Somente a acção de saída do sistema (*unregister*) pode levar à alteração desse ciclo de evolução.

b) Enquanto membro de um grupo, uma entidade elementar pode assumir diferentes estados lógicos, consoante a sua visibilidade e acessibilidade aos outros membros do grupo (ver figura 4.7). Para permitir modelar situações em que a entidade não está acessível devido a factores físicos, tal como por exemplo, a falta de bateria no dispositivo móvel que suporta o programa da entidade, ou devido a falta de conectividade do ponto de acesso da entidade à infraestrutura de rede, considerou-se um estado lógico "desligada" (OFFLINE). No estado OFFLINE, uma entidade não pode interagir com os outros membros do grupo a que pertence, seja para emitir, seja para receber mensagens ou eventos ou aceder ao espaço partilhado. Em particular, todas as comunicações de mensagens ou eventos que lhe sejam dirigidas enquanto nesse estado, perdem-se, isto é não serão posteriormente entregues automaticamente à entidade quando esta se voltar a ligar. No entanto a aplicação pode tratar de as entregar, por exemplo através da gestão de um arquivo (*log*) persistente que guarde a história das interacções ocorridas. No estado OFFLINE a entidade também não tem acesso ao estado partilhado do grupo. O estado OFFLINE também pode modelar situações controladas explicitamente pela entidade, através da invocação da primitiva *set_mode*, já anteriormente referida. Neste caso, bem como no caso de se atingir o estado de OFFLINE por falta de conectividade, o programa associado à entidade pode continuar em execução, desde que as acções desencadeadas se possam completar sem necessitar de interacção com o seu ambiente.

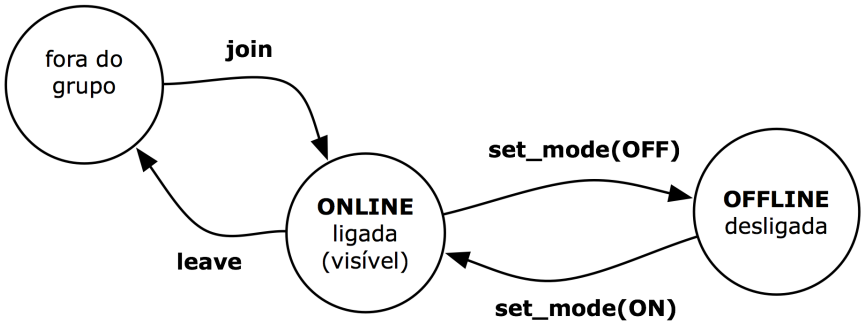

Figura 4.7: Esquema de transição de estados, da entidade, enquanto membro de um grupo.

Ao filiar-se num grupo, a entidade fica automaticamente no estado "ligada" (ONLINE) ficando "visível" aos restantes membros do grupo. De salientar que, todas as entidades, independentemente do seu estado (ONLINE ou OFFLINE), são consideradas como membros a partir do momento da sua filiação (*join*) no

mesmo. As entidades OFFLINE no entanto não fazem parte da visão do grupo, para efeitos do modelo de consistência como já foi anteriormente referido.

4.5 Interacções entre entidades

Este modelo tem como principal objectivo facilitar a estruturação/organização e modelação das interacções entre entidades. Estas entidades podem ser entidades elementares ou grupos, sendo a ênfase dada à comunicação no contexto de grupos de utilizadores e à forma como essa informação é divulgada pelos membros dos grupos.

Os mecanismos de interacção disponibilizados no modelo podem ser divididos nas seguintes categorias de comunicação:

- Directa: comunicação entre entidades (elementares ou grupos) em que o emissor nomeia explicitamente o receptor;
- Eventos: mecanismo associado à comunicação por eventos, que efectua a notificação assíncrona de uma mensagem para um conjunto de entidades que subscreveu esse evento no grupo;
- Espaço partilhado: forma de comunicação entre membros de um mesmo grupo que recorre à utilização de um conceito de memória partilhada (neste caso um espaço de tuplos).

4.5.1 Comunicação directa

A comunicação directa refere-se ao estabelecimento de uma comunicação explícita entre duas entidades, em que a entidade emissora envia explicitamente para um dado destinatário, uma mensagem, através da invocação da primitiva $send()$. O destinatário, por sua vez, recebe e processa a mensagem que lhe foi enviada de acordo com o método invocado por esta, que é explicitado na mensagem. As entidades têm disponíveis dois mecanismos distintos para o estabelecimento da comunicação directa que são apresentados de seguida.

Método explícito do receptor

Através deste mecanismo, o emissor invoca explicitamente um método da interface do receptor (entidade), responsável pela recepção e tratamento da mensagem.

$$\textbf{send}(\underline{in} : entidade\_emissora\_id, entidade\_receptora\_id, metodo\_recepcao, mensagem)$$

Nesta situação o destinatário (*entidade_receptora*), por cada método invocado, desencadeia a execução do método de recepção indicado (*metodo_recepção*) que é o responsável por efectuar o processamento da mensagem enviada pela entidade emissora.

A definição e implementação de cada método é da responsabilidade de quem desenvolve a aplicação, sendo estes associados à entidade quando da sua criação/definição, e fazem parte da sua interface. O conjunto de métodos que a entidade disponibiliza para a interacção é uma das características que definem a interface oferecida por cada entidade e deve ser do conhecimento do elemento emissor.

Este mecanismo permite a comunicação directa entre quaisquer duas entidades do sistema. É uma forma de interacção não bloqueante, não existindo confirmação de recepção por parte da entidade destinatária. Como no caso em que a entidade emissora pretende efectuar uma comunicação directa com uma entidade que se encontra OFFLINE, nesta situação a acção de *send* não fica a aguardar pela recepção.

Fila de mensagens no receptor

Com este mecanismo, é activado no destinatário, um método genérico que trata da recepção da mensagem colocando-a numa estrutura interna ao receptor (fila de mensagens), que pode ser acedida através da invocação, por parte do receptor, da primitiva de leitura explícita *receive*. No caso do destinatário ser um grupo a mensagem é colocada na fila global do grupo.

A acção de envio directo de uma mensagem é desencadeada na sequência de uma acção de *send* por parte do emissor.

$$\textbf{send}(\underline{in} : entidade\_emissora\_id, entidade\_receptora\_id, tipo\_notificacao, mensagem)$$

Quando o destinatário é um grupo, a mensagem é processada por este através do representante de grupo. A forma como são "avisados" os membros do grupo depende do tipo de notificação especificado pelo emissor, através do parâmetro $tipo\_notificacao$. A divulgação da chegada de mensagem pelos membros de um grupo pode ter diferentes comportamentos, de acordo como o tipo indicado na invocação da primitiva $send()$. São consideradas as seguintes políticas de divulgação:

PASSIVE: neste caso a mensagem é dirigida somente ao representante do grupo, que a processa e coloca na fila de mensagens global do grupo, não sendo esta informação difundida aos membros do grupo;

NOTIFY: publica um evento de aviso[16] de chegada de mensagem ao grupo sendo, neste caso, da responsabilidade de cada membro desencadear o processo de leitura da mensagem;

SPREAD: publica um evento[17], que desencadeia um processo de notificação aos membros do grupo, através do qual a mensagem é difundida pelo grupo.

[16]Este evento de chegada de mensagem é um dos eventos de sistema (MESSAGE_EV). Este tipo de eventos é por convenção subscrito por todas as entidades do sistema.

[17]Este é também um evento de sistema (MESSAGE_SPREAD_EV).

Na situação em que o destinatário é uma entidade elementar, a colocação de uma mensagem na sua fila local de mensagens desencadeia um evento de aviso de chegada de mensagem, não sendo processado, neste caso, o argumento $tipo\_notificacao$.

Para efectuar a leitura de mensagens, o receptor tem à sua disposição a primitiva *receive*(), que permite a qualquer fluxo de execução de membros do grupo, ir "buscar" explicitamente uma mensagem à sua fila local de mensagens ou à fila global do grupo.

No caso da leitura da mensagem da sua própria fila de mensagens a primitiva de *receive*() toma a forma:

receive($\underline{in} : tipo; \underline{out} : mensagem$)

Esta primitiva desencadeia a remoção de uma mensagem da fila utilizando uma estratégia FIFO.

No caso de leitura efectuada no contexto do grupo a primitiva *receive*() pode ser invocada por qualquer membro do grupo tratando-se de uma leitura não destrutiva, permanecendo as mensagens na fila global do grupo até expirar um certo *timeout*. Neste caso a primitiva *receive*, necessita dos seguintes parâmetros:

receive($\underline{in} : tipo, grupo\_id, identificador\_mensagem, filtro; \underline{out} : mensagem$)

- *tipo*: de acordo com este parâmetro a primitiva *receive*() pode ter um comportamento bloqueante ou não:

  BLOCK: bloqueante, fica a aguardar a disponibilidade de mensagem;

  NO_BLOCK: não bloqueante, se não existir mensagem para ler, retorna de imediato com *mensagem* vazia;

- *mensagem*: conteúdo de dados a ser entregue;

- $filtro$: permite efectuar uma leitura de mensagens, que obedeçam às definições do filtro, ou seja pode ser efectuado um processamento das mensagens de modo a aceder somente a determinado tipo de mensagens[18]. Indicando este filtro um valor que identifica o tipo de mensagem que se pretende ler. Este parâmetro assume significado no contexto de grupo, dado que, no caso das entidades elementares todas as mensagens que lhe são explicitamente dirigidas têm que ser sempre processadas por esta;

- $identificador\_mensagem$: este parâmetro permite identificar inequivocamente uma dada mensagem. Este valor é conhecido pelos membros do grupo após ter sido enviada uma mensagem (*send*()) com NOTIFY, sendo necessário para que os membros consigam ter acesso à mensagem sobre a qual foram notificados;

[18] As mensagem podem obedecer a um determinado formato no qual podem ser indicados, por exemplo, os diferentes tipos de mensagens existentes numa dada aplicação.

- *grupo_id*: identificação do grupo, necessária quando a leitura é efectuada da fila global do grupo, dado que as entidades podem efectuar leituras da fila de mensagens ou dos grupos a que pertencem. No caso do grupo, a leitura é efectuada da sua fila global, onde o representante do grupo colocou a mensagem. Note-se que a fila global do grupo só pode ser acedida por membros do próprio grupo, não sendo visível do exterior.

4.5.2 Eventos

Este mecanismo de comunicação suporta a difusão de eventos para um grupo de entidades. Esta comunicação é baseada num modelo de eventos, tal que a ocorrência de um evento permite a difusão de informação para os membros de um grupo que tenham previamente subscrito esse tipo de evento. O principal motivo da escolha de um modelo de eventos prende-se com o facto de um grande número de situações que se pretendem modelar, envolverem elementos de natureza essencialmente reactiva e assíncrona, que devem responder a alterações ocorridas dinamicamente. Os eventos são suportados por um esquema de subscrições e de notificações, como é típico nestes sistemas [EFGK03, TR05].

As entidades produtoras de informação publicam eventos, enquanto as entidades interessadas na informação sinalizam o seu interesse através de uma subscrição do tipo de evento, que é registada pelo sistema. As entidades subscritoras de um dado tipo de evento recebem uma notificação sempre que ocorrer um evento desse tipo, sendo então executado um método de atendimento especificado e que foi configurado para esse evento, quando da subscrição.

Um evento gerado possui, genericamente, o seguinte formato:

$evento(id\_produtor, msg\_ev)$

- *id_produtor*: identificação de entidade que desencadeou o evento;
- *msg_ev*: mensagem de evento desencadeada.

A ocorrência de um evento desencadeia um processo de notificação, onde a informação que é difundida para os membros do grupo ou seja o conteúdo de *msg_ev*, possui o formato:

$< tipo\_evento, atribs >$

- $tipo\_evento$: tipo de evento, corresponde a uma identificação que distingue o evento, de entre o conjunto de tipos reconhecidos, ou seja, registados no sistema;
- $atribs$: lista de atributos referentes ao evento, podendo ter a configuração, $< nome, valor >$, onde:

– *nome*: nome do atributo;

– *valor*: valor(es) assumidos pelo atributo.

Este tipo de interacção entre entidades é particularmente importante no âmbito de comunicação com um grupo (comunicação 1-N), ou seja, quando se pretende difundir a mesma informação para um conjunto de entidades de um mesmo grupo e ter uma reacção da parte dos seus membros. Este mecanismo de comunicação é utilizado no caso da comunicação directa (*send*) quando o destinatário é um grupo e se pretende avisar os seus membros da chegada de informação, sendo estes notificados através de um evento interno do sistema.

Existem duas classes distintas de eventos, os eventos de sistema e os eventos de aplicação. Os primeiros são definidos e utilizados ao nível do próprio sistema, servem como forma de realização de algumas das primitivas definidas no modelo e são automaticamente subscritos por todas as entidades quando estas são registadas. Os segundos são eventos definidos e configurados pela aplicação, ou seja, é da responsabilidade desta a definição do comportamento/método que deve ser desencadeado quando da ocorrência do evento, bem como a estrutura de notificação do próprio evento (*atribs*). Os membros que pretendem ser notificados da ocorrência de um tipo de evento de aplicação deverão subscrevê-lo e ter acesso à rotina de atendimento[19] que será executada sempre que esse evento ocorrer.

Todos os eventos são disseminados somente dentro de um grupo, ou seja, somente os membros têm acesso aos eventos definidos para o grupo. As entidades exteriores ao grupo não podem subscrever nem ter acesso aos serviços de notificação de eventos específicos do grupo. Como já foi referido, cada membro do grupo pode seleccionar os tipos de eventos cujas notificações pretende subscrever.

O modelo de eventos é suportado pelas primitivas:

- *advertise / unadvertise / ckeck_adv*;
- *publish*;
- *subscribe / unsubscribe*.

Estas primitivas permitem estabelecer e gerir as interacções, através de eventos, dos membros de um grupo.

Anúncio de novo tipo de evento

advertise($\underline{in}$: $entidade\_id, grupo\_id, tipo\_advertise, tipo\_evento, info\_evento$)

Esta primitiva anuncia, no grupo, que determinada entidade do grupo pode vir a "publicar" um evento, ou seja, pode ser geradora desse tipo de evento. Ao executar

[19] Associada ao tipo de evento quando da subscrição.

esta primitiva, é efectuada uma actualização de uma estrutura de suporte a eventos no grupo. Esta estrutura, que pode ser consultada por todos os membros, é gerida pelo representante de grupo, uma vez que este é o responsável pela gestão das comunicações no grupo. A estrutura contém informação referente ao evento e à lista dos membros que o subscreveram; no caso de se tratar de um evento de sistema esta lista coincide com a lista de membros do grupo.

Cada tipo de evento tem associada uma rotina de tratamento, a que as entidades subscritoras têm que ter acesso e que é executada por estas quando recebem a notificação da ocorrência do tipo de evento.

- *entidade_id*: identificação da entidade que será a produtora dos eventos do tipo indicado;
- *grupo_id*: identificação do grupo;
- *tipo_advertise*: tipo de anúncio que irá ser desencadeado e que indica qual a política de divulgação da ocorrência do evento, para os membros do grupo:

  PASSIVE: esta opção selecciona uma política em que é efectuado um anúncio passivo do novo tipo de evento, ou seja, é da responsabilidade dos possíveis interessados consultar, através da primitiva *check_adv*, que tipos de eventos existem disponíveis no grupo;

  ACTIVE: esta opção selecciona uma política em que é efectuado um anúncio activo do tipo de evento, ou seja, é desencadeado um evento de anúncio de evento sendo notificados todos os membros do grupo da ocorrência de um novo *advertise*, por parte de uma entidade do grupo;

- *tipo_evento*: identificador do tipo de evento, atribuído pela entidade anunciante; a primitiva *advertise* lida somente com eventos de aplicação dado que os eventos de sistema são sempre, à partida, "conhecidos" por todos;
- *info_evento*: informação associada ao tipo de evento, com descrição da estrutura da informação associada ao evento, ou seja a lista de atributos com valores não instanciados ($< nomeatributo, \_ >$).

Anúncio de remoção de um tipo de evento

unadvertise($\underline{in}$: $entidade\_id, grupo\_id, tipo\_evento$)

Ao executar esta primitiva, a entidade sinaliza que deixa de ser produtora de eventos deste tipo. Sendo também removido/desactivado este tipo de evento das estruturas de suporte de eventos no representante do grupo.

- *entidade_id*: identificação da entidade produtora do evento;

- *grupo_id*: identificação do grupo;

- *tipo_evento*: tipo de evento (identificador do evento).

Consulta de tipos de eventos disponíveis

check_adv($\underline{in}$: *grupo_id*; $\underline{out}$: [*tipo_evento*, *entidade_id*, *metodo_atend*]

Os membros do grupo podem consultar informação (na estrutura de suporte de eventos do grupos) sobre os diversos tipos de eventos de aplicação que podem ocorrer dentro do grupo[20], através da primitiva *check_adv*() Assim, podem efectuar uma consulta de tipos de eventos que foram anunciados via *advertise*() e as entidades que os produzem, no grupo indicado. Esta informação encontra-se definida e é gerida pelo representante de grupo. Ao executar esta primitiva, é devolvida a lista de tipos dos eventos anunciados nesse grupo.

- *grupo_id*: identificação do grupo;

- [*tipo_evento*, *entidade_id*, *metodo_atend*]: lista com os tipos de eventos que foram anunciados para este grupo, as respectivas entidades produtoras e métodos de atendimento declarados.

Subscrição de um tipo de evento

subscribe($\underline{in}$: *grupo_id*, *tipo_evento*, *metodo_atend*)

Os membros do grupo podem subscrever tipos de eventos de aplicação que foram anunciados no grupo. A subscrição de um tipo de evento, por parte de uma entidade elementar, configura a sua estrutura interna de suporte de eventos, associando ao tipo de evento o método de atendimento indicado. Esta configuração permite desencadear o processo de atendimento, quando a entidade é notificada da ocorrência desse tipo de evento.

- *grupo_id*: identificação do grupo;

- *tipo_evento*: identificação do tipo de evento que pretende subscrever;

- *metodo_atend*: identificação do método de atendimento a ser executado pela entidade subscritora, quando ocorre esse evento.

[20] Os tipos de eventos aqui referidos são os que foram anunciados *advertise*().

Remoção da subscrição de um tipo de evento

unsubscribe($\underline{in}$: $grupo\_id, tipo\_evento$)

Esta primitiva desencadeia o processo de cancelamento da subscrição de um tipo de evento, anteriormente subscrito pela entidade nesse grupo.

- *grupo_id*: identificação do grupo;
- $tipo\_evento$: identificação do tipo de evento que pretende deixar de subscrever.

Publicação de um evento

publish($\underline{in}$: $grupo\_id, evento$)

Esta primitiva é invocada pela entidade que pretende notificar os membros do grupo da ocorrência de determinado tipo de evento. O responsável pela notificação dos membros do grupo informa-os da ocorrência do evento. Estes, de acordo com a sua configuração de eventos subscritos, desencadeiam o comportamento respectivo (definido no método de atendimento associado ao tipo de evento).

- *grupo_id*: identificação do grupo;
- *evento*: evento ocorrido, que obedece ao formato já referido anteriormente.

Este mecanismo de comunicação permite efectuar a difusão de informação dentro de um dado grupo . A difusão é efectuada para todos os membros ligados do grupo e que subscreveram este tipo de evento. No caso dos membros não pretenderem receber um determinado tipo de evento, devem invocar a primitiva *unsubscribe*() que desactiva o atendimento de eventos desse tipo, actualizando a estrutura de suporte de eventos,

4.5.3 Espaço partilhado

A comunicação através de um espaço partilhado permite realizar formas de interacção indirectas entre os membros de um grupo, nas quais os produtores de informação não necessitam de designar explicitamente os consumidores da mesma. O espaço partilhado, dada a atomicidade das primitivas que o manipulam, permite também expressar formas de coordenação entre os membros do grupo. Por outro lado, este espaço pode também ser visto como um repositório (volátil ou persistente), que cada aplicação pode utilizar para registar a história e a evolução das interacções entre os membros do grupo, incluindo dados e/ou referências para a informação relevante à aplicação.

Este conceito de espaço partilhado, uma vez associado ao contexto de um grupo, permite modelar situações de partilha de informação de uma forma estruturada e selectiva, explorando ao mesmo tempo a localidade e a afinidade de objectivos entre os membros de um grupo.

Por outro lado, a comunicação através do espaço partilhado pode ser associada ou não, conforme as necessidades de cada aplicação, ao mecanismo de notificação baseado em eventos.

Na concepção das primitivas manteve-se a semântica do modelo Linda [CG89, Gel85] (ao suportar operações equivalentes a *in*, *rd* e *out*) e procurou-se disponibilizar outras operações para aumentar a flexibilidade da pesquisa e consulta.

Através deste mecanismo, os produtores de informação colocam os dados no espaço partilhado do grupo, podendo também proceder à notificação dos membros do grupo. A acção de consulta dos dados é sempre da exclusiva responsabilidade dos possíveis interessados (consumidores).

Devido à multiplicidade de possíveis configurações de dados, a opção aqui tomada, em termos de formato de tuplo manipulado, foi a de ser genérica, deixando às aplicações a responsabilidade de definir o formato da informação colocada no espaço partilhado do grupo.

Em contraste com o modelo Linda atribui-se, neste modelo, uma interpretação dos campos que constituem os tuplos de dados no espaço partilhado, não comprometendo no entanto a generalidade das primitivas propostas. O tuplo de dados possui o seguinte formato:

$$< produtor\_id, tipo\_acesso, dados >$$

Correspondendo estes campos a:

- *produtor_id*: identificador do produtor dos dados;
- *tipo_acesso*: tipo de acesso permitido, foram definidos dois tipo de acesso já que é assumido (como semântica de grupo), que toda a informação deve poder ser acedida por todos os seus membros[21]:

  PUBLIC: acesso geral a todos os membros do grupo;

  PRIVATE: acesso permitido somente ao produtor;

- *dados*: lista de parâmetros; a constituição desta lista assim como o número de parâmetros e seus tipos são definidos pela aplicação.

O conjunto de primitivas disponíveis para escrita e leitura do espaço partilhado foram, tal como já foi mencionado, inspiradas no modelo Linda e mantêm a sua atomicidade. Estas primitivas permitem efectuar a gestão dos dados comuns do grupo, que são mantidos através do seu espaço partilhado, sendo responsáveis por desencadear o processo de actualização e consulta de dados no contexto de partilha do grupo. As primitivas foram desenvolvidas em função do tipo de utilização que as aplicações poderão necessitar na gestão e coordenação dos seus dados, sendo dos seguintes tipos:

[21] Em situações em que se pretende que somente alguns dos membros do grupo tenham acesso à informação deve repensar-se a formação do grupo, por exemplo através da criação de mais do que um grupo.

- primitiva de colocação de dados no espaço: $update()$;
- primitivas de pesquisa (leitura não destrutiva) de dados: $find()$, $consult()$;
- primitiva de leitura de dados: $get()$.

Actualização/escrita de dados no espaço

update($\underline{in}$: $entidade\_id, grupo\_id, tipo\_acesso, tipo\_notificacao, dados$)

Primitiva (equivalente ao *out* no modelo Linda) que actualiza o espaço de dados partilhado, colocando/escrevendo um novo tuplo no espaço.

Com base nos parâmetros de chamada desta primitiva, é construído um tuplo que é inserido no espaço partilhado do grupo. Os parâmetros especificados definem também o comportamento, face aos restantes membros do grupo, desencadeado pela alteração dos dados.

Os argumentos da primitiva são:

- *entidade_id*: identificador do produtor de informação;
- *grupo_id*: identificador do grupo;
- $tipo\_acesso$: tipo de acesso, o produtor pode optar por não tornar imediatamente acessíveis os dados. Nesse caso deve indicar que os dados são PRIVATE ou seja, esta informação só pode ser consultada pelo produtor da mesma. Para que os dados sejam visíveis ao grupo, deve colocá-los como PUBLIC;
- $tipo\_notificacao$: aqui o produtor dos dados pode indicar se pretende que seja efectuada uma notificação (aviso) de actualização, aos membros do grupo:

  NOTIFY_UPD: quando de pretender notificar todos os membros do grupo, que foi efectuada uma operação de actualização do espaço[22];

  NO_NOTIFY_UPD: quando não se pretende informar, os membros do grupo, que foi efectuada uma actualização do seu espaço.

- *dados*: conteúdo da informação, definido pela aplicação, a ser colocada no espaço comum do grupo.

O tuplo colocado no espaço é construído com base nos argumentos de chamada da primitiva $update()$.

No caso de grandes volumes de dados[23], a aplicação pode optar por não os colocar directamente neste espaço partilhado, efectuando o seu armazenamento no sistema de informação (SI), sendo somente a referência de acesso aos dados (no SI) que é inserida

[22]Esta notificação é desencadeia um evento de sistema - UPDATE_EV.
[23]De um modo geral toda a informação multimédia.

no tuplo. Em termos da estrutura da informação, a aplicação pode definir, por exemplo, a seguinte lista de parâmetros: referência de acesso aos dados; formato de ficheiro de informação (por exemplo jpeg, mpeg, bmp); data de criação; conjunto de etiquetas (*tags*) que categorizem o conteúdo dos dados. O conjunto de etiquetas pode ser um subconjunto de palavras chave pré-definidas pela aplicação, que permitem atribuir um significado semântico aos dados. Com base na categorização dos dados, podem ser definidos, pela aplicação, critérios de pesquisa sobre o espaço partilhado.

Para modificar o tipo de permissões de acesso aos dados (privados ou públicos) foi definida a primitiva auxiliar:

set_access($\underline{in}$: $produtor\_id, grupo\_id, tipo\_acesso, dados$)

Esta primitiva permite a alteração do parâmetro de acesso aos dados:

- $tipo\_acesso$ =PRIVATE os dados são privados;
- $tipo\_acesso$ =PUBLIC os dados podem ser acedidos por todos os membros do grupo.

Somente o produtor dos dados pode alterar a configuração de acesso aos mesmos. O parâmetro *dados* deve corresponder exactamente aos dados do tuplo cujo tipo de acesso se pretende modificar.

Pesquisa de informação no espaço

Para a pesquisa de informação, ou seja, uma leitura não destrutiva do espaço partilhado, foram definidas primitivas que permitem efectuar a leitura de informação de forma associativa. Essas primitivas são descritas de seguida.

find($\underline{in}$: $entidade\_id, grupo\_id, produtor\_id, sinc, lista\_valores$; $\underline{out}$: num_tuplos)

A primitiva $find()$ efectua a pesquisa de informação no espaço partilhado do grupo indicado (*grupo_id*).

Com base nos parâmetros de chamada, é elaborado um tuplo, com formato idêntico ao especificado quando do *update* e a partir do qual é efectuada a pesquisa, ou seja, os tuplos do espaço que satisfaçam o padrão de pesquisa são lidos do espaço partilhado do grupo, para uma estrutura local a cada entidade, cuja informação pode ser consultada através de uma primitivas de consulta ($consult()$), que é descrita a seguir. A estrutura local, onde são armazenados os tuplos que satisfizeram as condições da pesquisa, mantém o estado após a execução de cada operação de $find()$, sendo da responsabilidade da aplicação gerir a forma como é efectuada a consulta dos tuplos.

Esta primitiva permite efectuar uma leitura (não destrutiva) de todos os tuplos do espaço que satisfaçam os critérios de pesquisa, distinguindo-se da operação de leitura do modelo Linda, pelo facto de possibilitar a leitura simultânea de vários tuplos.

Os argumentos definidos para esta primitiva definem os critérios da pesquisa no espaço de tuplos:

- *entidade_id*: identificador da entidade que pretende efectuar a pesquisa;
- *grupo_id*: identificação do grupo sobre o qual se deseja efectuar a pesquisa;
- *produtor_id*: identificador da entidade que escreveu os dados (produtora); caso não seja especificada a entidade produtora, significa que a pesquisa não entra com essa condição na pesquisa;
- *sinc*: indicação de se a pesquisa é bloqueante ou não:

  BLOCK: pesquisa com semântica bloqueante, fica a aguardar que exista pelo menos um tuplo que verifique as condições;

  NO_BLOCK: pesquisa com semântica não bloqueante, no caso de não existir nenhum tuplo que satisfaça as condições retorna com *num_tuplo* com o valor 0.

- *lista_valores*: campo com lista de valores para uma pesquisa personalizada sobre o espaço partilhado, sendo da responsabilidade da aplicação garantir que esta lista possui o mesmo formato dos dados, conforme definido quando da escrita no espaço de tuplos (*update*);
- *num_tuplos*: número de tuplos do espaço partilhado, que satisfazem as condições da pesquisa ou seja número de tuplos do espaço que são compatíveis com o tuplo de pesquisa, no caso de nenhum tuplo do espaço verificar as condições e a semântica de leitura ser não bloqueante toma o valor 0.

Como já foi referido, é da responsabilidade da aplicação a definição do formato do tuplo de dados e dos valores que cada um dos campos do tuplo pode tomar.

Por exemplo, se for definido, pela aplicação, um formato para os dados, do tuplo de dados, com a estrutura: $[tipo\_conteudo, tema, local, data]$, pode ser criado um tuplo de pesquisa com a seguinte configuração, em que todos os campos se encontram instanciados: $[imagem, edificio, cidade, 21/04/2007]$. Também, obedecendo ao mesmo formato, podem ser definidos tuplos com campos não instanciados, como por exemplo: $[imagem, edificio, -, -]$. Neste caso são lidos do espaço todos os tuplos que satisfaçam os dois primeiros parâmetros e que podem ter qualquer valor nos restantes campos.

Como já foi referido, os tuplos lidos do espaço são colocados numa estrutura (tabela) interna de cada entidade, que pode ser posteriormente acedida por esta através da primitiva $consult()$. Sempre que é efectuada uma pesquisa ($find()$) sobre o espaço partilhado esta estrutura é reconstruída, não sendo no entanto garantido pelo modelo que, entre a operação de pesquisa e a consulta, não tenha ocorrido uma alteração dos dados no espaço partilhado e/ou no sistema de informação, por parte de outra entidade concorrente.

Consulta local

consult($\underline{in}$: num; $\underline{out}$: $tuplo\_dados$)

Esta primitiva permite efectuar a leitura de um dado tuplo da estrutura interna, que foi actualizada pelo resultado da operação de pesquisa ($find()$), onde cada tuplo é identificado por um valor (*num*). Do ponto de vista de uma aplicação a utilização das primitivas de pesquisa ($find()$) e de consulta local ($consult()$), estão directamente relacionadas e devem ser utilizadas em conjunto.

- *num*: identificador do tuplo, que corresponde à sua posição na estrutura interna (tabela), e não deve exceder[24] o valor *num_tuplos* retornado após a execução da primitiva $find$, responsável pela actualização da estrutura.
- $tuplo\_dados$: tuplo de dados lido da estrutura interna e que obedece ao formato definido.

Consulta de um tuplo do espaço

consult($\underline{in}$: $entidade\_id, grupo\_id, produtor\_id, sinc, list\_valores$; $\underline{out}$: $tuplo\_dados$)

Esta versão da primitiva de consulta efectua a leitura de um tuplo do espaço partilhado e segue a semântica definida pelo modelo Linda para a leitura sem remoção *rd*. Tal como na primitiva de pesquisa ($find()$), é construído um tuplo a partir dos argumentos de invocação da primitiva, com base no qual é efectuada a pesquisa no espaço partilhado. Também como aquela pode ter um comportamento bloqueante ou não bloqueante que é configurado a partir do parâmetro de entrada *sinc*. No caso não bloqueante, na situação em que não foi encontrado nenhum tuplo no espaço que satisfaça as condições de consulta, retorna tuplo "vazio" em $tuplo\_dados$.

É devolvido um tuplo, o primeiro encontrado, que obedeça aos parâmetros de pesquisa. Tal como na proposta do modelo Linda, não é garantido que consultas consecutivas não retornem o mesmo tuplo, já que o espaço de tuplos não é ordenado.

- $entidade\_id$: identificação da entidade do grupo que pretende consultar os dados. Só pode consultar dados públicos e os seus dados privados;
- $grupo\_id$: identificação do grupo sobre o qual se deseja efectuar a consulta;
- $produtor\_id$: identificação da entidade que escreveu os dados (produtora) no espaço;
- *sinc*: indicação se a consulta é bloqueante ou não:

[24] No caso de o valor *num* ser maior do que o número de tuplos lidos quando da última pesquisa, o tuplo de dados retornado será vazio.

BLOCK: consulta com semântica bloqueante;

NO_BLOCK: consulta com semântica não bloqueante;

- *list_valores*: campo com lista de valores para uma pesquisa personalizada sobre o espaço partilhado, devendo a aplicação garantir que esta lista possui o mesmo formato dos dados, definido quando da escrita no espaço de tuplos (*update*);
- *tuplo_dados*: tuplo retornado pela consulta do espaço.

Remoção de dados

Tal como no caso do *consult*(), foram definidas duas versões para efectuar a extracção de tuplos do espaço, que são descritas de seguida.

get($\underline{in} : num; \underline{out} : tuplo\_dados$)

Nesta versão é efectuada a leitura do tuplo identificado por *num*, da tabela construída pela pesquisa (*find*()), desencadeando a primitiva de *get* as seguintes acções:

- acesso à tabela de tuplos com base no parâmetro *num* e obtenção do tuplo;
- remoção da entrada na tabela;

O tuplo removido da estrutura local é retornado em *tuplo_dados*, no caso de não ter sido possível efectuar a remoção do tuplo, por não existir nenhum tuplo na posição *num*[25], retorna um tuplo de dados "vazio".

get($\underline{in} : entidade\_id, grupo\_id, produtor\_id, sinc, lista\_valores; \underline{out} : tuplo\_dados$)

Para efectuar a leitura com remoção de dados do espaço partilhado foi definida a primitiva *get*(), equivalente à primitiva *in* do modelo Linda. Esta primitiva é semelhante à de *consult*(), excepto que efectua a remoção do tuplo que é compatível com tuplo de pesquisa e que foi construído com base nos parâmetros de entrada da primitiva.

Foram definidas duas semânticas de leitura para esta primitiva: bloqueante, em que fica a aguardar que exista no espaço um tuplo que satisfaça as condições, para poder proceder à sua remoção; não bloqueante, se não existir no espaço nenhum tuplo compatível retorna de imediato com tuplo de dados "vazio".

- *entidade_id*: identificação da entidade (membro do grupo) que pretende remover os dados[26];

[25] Não é garantida a consistência entre os dados da estrutura local e o espaço partilhado.

[26] As entidade só podem remover os seus dados próprios (definidos como PRIVATE) ou dados públicos do grupo (definidos como PUBLIC).

- *grupo_id*: identificação do grupo sobre o qual se deseja efectuar a leitura;
- *produtor_id*: identificador da entidade (produtora) que escreveu os dados no espaço;
- *sinc*: indicação de se a leitura é bloqueante ou não

  BLOCK: leitura com semântica bloqueante;

  NO_BLOCK: leitura com semântica não bloqueante;

- *lista_valores*: campo com lista de valores para construção do tuplo de pesquisa;
- *tuplo_dados*: tuplo retornado pela leitura (com remoção) do espaço partilhado.

Políticas de notificação de actualização do espaço

O mecanismo de interacção entre membros baseado no espaço partilhado do grupo, pode seguir duas políticas:

- Colocação de dados com notificação automática aos restantes membros do grupo, relativamente à alteração observada;
- Colocação de dados sem notificação, ou seja os membros podem aceder à informação, mas não foram notificados da alteração que ocorreu no estado do espaço partilhado.

Para o desenvolvimento deste mecanismo de notificação, recorreu-se à utilização de eventos como forma de anunciar as alterações aos restantes membros. Os processos de gestão associados ao espaço partilhado, têm definido um método para lidar com a ocorrência de eventos de actualização de dados. Ao ser executada uma primitiva que provoque uma modificação do espaço partilhado do grupo, com opção de NOTIFY_UPD, é desencadeado um processo de notificação ao membros do grupo.

Esta notificação é efectuada através do mecanismo de eventos anteriormente descrito. Os membros, ao filiarem-se num grupo subscrevem automaticamente este tipo de evento, passando a ser notificados quando for efectuada uma actualização[27] do espaço partilhado.

Na tabela 4.1, encontra-se o resumo das primitivas disponibilizadas pelo modelo MAGO, sendo apresentado na secção seguinte o esquema de utilização das mesmas.

[27]Isto, no caso de o elemento responsável pela actualização ter optado por *tipo_notificacao* = NOTIFY_UPD.

Primitivas do modelo			*parâmetros de entrada*	*parâmetros saída*
criação		*register*	nome_id, lista atributos	identificador
		unregister	identificador, nome_id	
gestão de grupo		*create*	entidade_id, nome grupo, tipo grupo, tipo acesso, max membros, [lista parâmetros]	identificador
		destroy	identificador, nome grupo, tipo aviso	
		join	entidade_id, grupo_id, info	entrada(valida)
		leave	entidade_id, grupo_id	saída(valida)
		membership	grupo_id	
		set_mode	entidade_id, grupo_id, modo	
comunicação	**directa**	*send*	entidade_snd_id, entidade_rcv_id, modo recepção, tipo, mensagem	
		receive	tipo	
		receive[2]	tipo, identificador mensagem, grupo_id, filtro	
	eventos	*advertise*	entidade_id, grupo_id, tipo advertise, tipo evento, info	
		unadvertise	entidade_id, grupo_id, tipo evento	
		check_adv	grupo_id	lista:[tipo evento, entidade_id]
		subscribe	grupo_id, tipo evento,metodo atendimento	
		unsubscribe	grupo_id, tipo evento	
		publish	grupo_id, evento	
	espaço partilhado	*update*	entidade_id, grupo_id, tipo acesso, tipo notificação, dados	tuplo id
		find	entidade_id, grupo_id, entidade_prod_id, sinc, lista valores	num tuplos
		consult	num	tuplo dados
		consult[2]	entidade_id, grupo_id, entidade_prod_id, sinc, lista valores	tuplo dados
		get	num	tuplo dados
		get[2]	entidade_id, grupo_id, entidade_prod_id, sinc, lista valores	tuplo dados

Tabela 4.1: Conjunto de primitivas base disponibilizadas no modelo MAGO

4.6 Modelação - primitivas do modelo

Nesta secção discutem-se com maior desenvolvimento as primitivas do modelo, com o objectivo de ilustrar, com pequenos exemplos em pseudo-código, a sua utilização e o formato de invocação.

4.6.1 Registo (*register*)

Pressupõe-se que uma entidade elementar, para fazer parte do sistema, deverá efectuar o seu registo, sendo-lhe atribuído um identificador pelo qual passa a ser conhecida no sistema. O registo corresponde à sua filiação como novo membro no grupo universal (GU), do qual todas as entidades do sistema fazem parte. No acto do registo, a entidade deverá fornecer os dados requeridos pela aplicação em particular, para além dos dados requeridos pelo modelo. Ao efectuar o registo, a entidade passa a estar integrada no grupo universal, podendo a partir daí interagir com as restantes entidades, conforme seja especificado pela aplicação.

```
...
{ pedido de nome (login) }
{ preencher campos de dados referentes à entidade }
{ preencher os diversos campos de dados característicos da aplicação }
name = { login utilizador, atribuído ao nível da aplicação }
info = { dados }
info_app = dados definidos pela aplicação
```

```
list_attribs = info + info_app

id = register(in: name, list_attrib; out: identif)

if (identif !=  null)
        // registo efectuado com sucesso
else
        // registo não foi possível de efectuar
...
```

Listagem 4.1: Exemplo de registo de um novo elemento

Ao efectuar o registo, é retornada uma identificação única, atribuída pelo sistema à entidade.

4.6.2 Criação de grupo (*create*)

A criação de um novo grupo pode ser desencadeada por qualquer entidade elementar registada no sistema. É da responsabilidade da aplicação definir tipos de critérios mais restritivos, já que o modelo permite a criação de grupos por parte de qualquer entidade registada. A criação de um grupo, independentemente do seu tipo (explícito ou implícito), é composta por três fases:

1. validação dos dados e características referentes ao grupo
2. execução da criação
3. confirmação da criação, com a respectiva actualização das estruturas de suporte de grupos

A criação de um grupo envolve, da parte do sistema, a criação da entidade representante desse grupo, que passa a ser a responsável pela gestão das interacções entre o exterior e o grupo, actuando como mediador das comunicações com o grupo e entre membros do grupo. Ao ser criado, são definidos diversos parâmetros de caracterização do grupo tais como a política de filiação, o tipo de mensagens e eventos que irá suportar, o número de membros e o tipo de grupo (explícito ou implícito).

Um exemplo de uma possível chamada da primitiva *create* de um grupo explícito:

```
...
{ aplicação valida nome_simbólico do grupo }
{ aplicação valida criador do grupo, analisando se este tem permissão para criação de
    grupo }

// define o tipo de grupo, e o conjunto de funcionalidades a ele associado
tipo_grupo = EXPLICIT

// neste caso para acesso basta que um dos membros do grupo valide o pedido de join
// note-se que o primeiro membro entra directamente
tipo_acesso =  BY_ONE
```

```
// permite numero máximo de 5 membros
max_membros = 5

valid = create(in: id_criador, nome_grupo, tipo_grupo, tipo_acesso, max_membros; out:
    group_id);

if ( valid != OK ) {
        // ERRO – não foi permitida a criação do novo grupo
}
if ( valid == OK ) {
        // grupo criado com sucesso com identificador atribuído (group_id),
        // a partir deste instante podem ser efectuados pedidos de filiação
}
...
```

Listagem 4.2: Criação de um grupo explícito (*create*)

Um exemplo de uma possível chamada da primitiva *create* de um grupo implícito:

```
...
{ aplicação valida nome_simbólico do grupo }
{ aplicação valida criador do grupo, analisando se este tem permissão para criação de
    grupo }

// define o tipo de grupo, e o conjunto de funcionalidades a ele associado
tipo_group = IMPLICIT
// acesso a todos os que pretendam
tipo_acesso =  ALL
// número ilimitado de membros
max_members = -1

[list_params] = { conjunto de regras que devem ser verificadas pelo membros }

validate = create(in: id_criador, nome_grupo, tipo_grupo, tipo_acesso, max_membros,
    list_params; out:  id_grupo);

if ( validate != OK ) {
        ERRO - não foi permitida a criação do novo grupo
}
if ( validate == OK ) {
        // grupo criado com sucesso
        // grupo criado com sucesso com identificador atribuído (group_id)
        // devido a ser grupo implícito primitiva create desencadeia evento de
        // filiação automática aos membros
        // registados e activos, que verifiquem os atributos da lista
        // e novos membros que se registem ou se tornem activos no sistema
        // recebem notificação para
        // filiação nos grupos implícitos entretanto criados
}
...
```

Listagem 4.3: Criação de um grupo implícito (*create*)

Note-se que, ao registarem-se no sistema, os membros passam a ter configurada a notificação de eventos de filiação, sendo da responsabilidade da implementação a gestão das entradas (e saídas) nos grupos implícitos.

A criação de um grupo implícito, que é caracterizado por um conjunto de características definidas ao nível da aplicação, desencadeia automaticamente o processo de filiação por parte dos membros, sem interferência explícita por parte do programa da entidade, ou seja, não existe uma interferência explícita, por parte da entidade, neste processo.

O processo de registo, assim como o de modificação do perfil de características[28] de entidade, desencadeiam um evento de verificação de compatibilidade com os grupos implícitos existentes.

O processo de criação e de manutenção de grupos implícitos está intimamente ligado com o sistema de informação definido para o caso particular de cada aplicação, tendo sido necessário definir um conjunto de métodos genéricos para a consulta e actualização dos dados que caracterizam cada entidade do sistema. Esses métodos são invocados/utilizados pelas primitivas do modelo que estão associadas à gestão dos grupos implícitos e dos perfis de utilizador.

4.6.3 Filiação num grupo (*join*)

Após o seu registo, uma entidade poderá optar por se filiar em grupos explícitos[29] existentes ou criar novos grupos, sendo a sua admissão dependente da política de acesso definida quando da criação de cada grupo. A primitiva de *join* é responsável por actualizar as estruturas de dados de suporte a grupos, da entidade que a invoca, com informação referente ao grupo em que se filiou, assim como actualizar as estruturas de suporte da configuração do grupo (geridas pelo representante do grupo)[30]. Entre estas estruturas salientam-se, por exemplo, as que descrevem a configuração dos tipos de eventos a que a entidade passa a ter acesso, após se tornar membro do grupo.

```
...
{ dado o nome simbolico, obter id_group }
{ preenche info requerida pela aplicação }
id = { identificador da entidade }

valid = join(in: id, id_group, info; out: addr_entity_group)

if ( valid != OK ) {
        // ERRO – não foi permitida a filiação no grupo
}
if ( valid == OK ) {
        // Filiação efectuada com sucesso
```

[28] Estas características de perfil de entidade são directamente dependentes da aplicação e do conjunto de atributos que foram definidos para as entidades elementares.

[29] Só tem sentido a chamada da primitiva de *join* em grupos explícitos, já que, a filiação nos grupos implícitos é dependente das características/propriedades da entidade e não depende de uma acção explícita de entrada por parte desta.

[30] As estruturas aqui referidas são vistas em detalhe no capítulo seguinte, onde é descrita a arquitectura de suporte.

```
            // addr_entity_group , acesso à entidade no grupo
    }
```

Listagem 4.4: Esquema de entrada num grupo (*join*)

A filiação no grupo obedece à política que foi configurada para o grupo, sendo aprovada somente se forem verificadas as condições necessárias de admissão.

4.6.4 Visualização da composição de grupos (*membership*)

Esta funcionalidade permite observar qual a composição actual de um grupo, ou seja, quais os membros que se encontram filiados nesse grupo.

```
    ...
    { aplicação valida nome_simbolico do grupo }

    // dado um grupo devolve em list os membros filiados e o seu status
    membership(in: id_group; out: lista )

    // visualização da lista de membros
    { com base na lista retornada, aplicação pode desencadear a visualização dos membros
        }
    ...
```

Listagem 4.5: Análise da constituição de um grupo (*membership*)

Cada grupo mantém uma estrutura dos seus membros correntes, assim como do seu estado (ONLINE, OFFLINE).

No caso dos grupos explícitos, é gerida uma lista pela entidade representante do grupo, com os identificadores dos membros aos quais foi concedida autorização de filiação. Para deixar de pertencer a essa lista, deverá ser executada uma invocação explícita de saída do grupo (*leave*) por parte do membro, caso contrário será considerado como pertencente ao grupo.

Na situação em que um membro do grupo fica temporariamente "desligado" do grupo (estado OFFLINE), é da responsabilidade do representante do grupo a actualização do estado desse membro na lista de membros do grupo. Quando se "ligar" (passar ao estado ONLINE) novamente é efectuada a reentrada do membro, sendo reconfiguradas as suas próprias estruturas de gestão de grupos, assim como do sistema e dos grupos a que este pertence. A detecção de interrupção na ligação e o pedido de invocação explícita de "reentrada" por parte do membro, fica a cargo da aplicação. Note-se que, ao reentrar no sistema, ou seja, ao passar do estado OFFLINE para ONLINE, os membros mantêm a filiação nos grupos onde validaram a sua entrada.

4.6.5 Troca directa de mensagens entre entidades (*send*/*receive*)

Ao registarem-se no sistema, as entidades passam a ter acesso a um conjunto de mecanismos de comunicação, sendo um deles a comunicação directa entre entidades ele-

mentares ou grupos. Algumas entidades podem ter definidos métodos especiais de recepção de mensagens, configurando a interface de recepção de mensagens que é invocada directamente quando do envio. No entanto, todas as entidades podem comunicar directamente com outras entidades, por invocação da primitiva de envio de mensagem (*send*()).

Ao enviar uma mensagem através do mecanismo *send* assume-se que o receptor deve efectuar uma acção explícita de recepção (*receive*). A primitiva *send*() tem um comportamento assíncrono não bloqueante. No entanto, por parte da aplicação, pode ser programada uma forma de sincronização entre o emissor e o receptor.

A comunicação através deste mecanismo requer o conhecimento do destinatário (quer seja uma entidade elementar ou grupo), já que tem que ser explicitado o seu identificador. Não é, no entanto, necessário que os dois interlocutores façam parte do mesmo grupo, uma vez que todas as entidades do sistema fazem parte do Grupo Universal, por definição do modelo.

Este mecanismo de interacção permite duas formas de interacção distintas, através da invocação explícita do método da interface da entidade receptora (dependente da aplicação), ou recorrendo a método genérico de recepção (inerente ao modelo).

Nesta forma de interacção, o emissor envia a mensagem que é tratada por um método do receptor que processa a mensagem: no caso de se tratar de uma entidade elementar, a mensagem deve ser lida da sua fila local de mensagens; no caso de ser um grupo, o representante deste desencadeia o processo de tratamento da mensagem, de acordo com o tipo de divulgação especificado.

A leitura de mensagens por parte do receptor, recorre à primitiva *receive*. Esta forma de recepção pode ser bloqueante, de acordo com o especificado na sua invocação e possui também mecanismos que permitem filtrar/selecionar o tipo de mensagem lida, de acordo com o tipo de mensagem ou o emissor.

Emissor: envio de mensagem que é colocada na fila de mensagens

```
 ...
message = 'Hello, i'm sending a message'
message_type = GENERIC
msg = {constrói a mensagem com base na message e no message_type}

// modo de divulgação da mensagem no caso de receptor ser um grupo
type = SPREAD

validate = send(in: id_sender, id_receiver, type, msg )

if ( validate !=OK )
        // Mensagem não enviada
if ( validate == OK )
        // Mensagem enviada
...
```

Listagem 4.6: Envio directo de mensagem para fila de mensagens do receptor (*send*)

Nesta emissão de mensagem através da primitiva *send*(), podem existir dois tipos de destinatários:

- grupo: neste caso, o representante de grupo é o responsável por processar a mensagem de acordo com a os parâmetros especificados na primitiva de *send*;
- entidade elementar: lê a mensagem da sua fila, através da primitiva *receive*(), ignorando o campo com o tipo dos argumentos.

Receptor: leitura de mensagem de fila de mensagens da própria entidade

```
...
msg = null
bloq = NO_BLOCK          // não bloqueante

receive(in: bloq; out: msg )

if ( msg == null )
        // Mensagem não disponível, fila vazia não existem mensagens para leitura
else
        // Mensagem é lida é removida da fila de mensagens
```

Listagem 4.7: Recepção assincrona de mensagem (*receive*)

Receptor: leitura de mensagem por parte de entidade, quando a mensagem foi enviada para um dos seus grupos

```
...
msg = null
[filter] = null                  // aberto a todo o tipo de mensagens
bloq = NO_BLOCK          // não bloqueante

{ aplicação valida nome_simbolico do grupo }

receive(in: bloq, id_message, id_group, filter; out: msg )

if ( msg == null )
        // Mensagem não disponível para leitura
else
        // Mensagem é lida da fila de mensagens do grupo
```

Listagem 4.8: Recepção assincrona de mensagem (*receive*)

Nesta situação de leitura explícita, a entidade recebeu previamente uma notificação (por parte do representante) de existência de mensagem disponível para leitura e com identificação desta (*id_message*). A identificação de mensagem é necessária neste caso pois a primitiva *receive*() vai efectuar uma leitura da fila global do grupo.

No caso de opção ser BLOCK em vez de NO_BLOCK, ao invocar a primitiva *receive*() o comportamento passa a ser bloqueante ou seja, fica a aguardar a "chegada" de uma mensagem que obedeça às condições especificadas nos parâmetros.

Troca de mensagens directa entre entidades utilizando um método da interface

Este mecanismo de comunicação é semelhante ao descrito anteriormente, mas neste caso recorre a um método definido na interface da entidade receptora.

Emissor

```
...
message = 'Hello, i'm using a method'
msg = { constrói a mensagem com base na message do emissor }

validate = send(in: id_sender, id_rcv, method_rcv, msg)

if ( validate != OK )
        // Mensagem não enviada
if ( validate == OK )
         // Mensagem enviada
...
```

Listagem 4.9: Envio directo de mensagem com utilização de método definido na interface do receptor (*send*)

Neste caso, do lado do receptor, é activado directamente o método *method_rcv*, que foi definido na interface de comunicação da entidade receptora a quem se destina a mensagem. A chamada do método desencadeia, na recepção, uma execução do método que processa a mensagem enviada, sendo este responsável por efectuar a leitura e o processamento da mensagem.

4.6.6 Gestão de eventos (difusão de mensagens para o grupo)

Os eventos permitem a difusão de mensagens para todos os membros de um grupo, de forma assíncrona, desencadeando a execução de um método de atendimento. Existem duas categorias de eventos distintos: os eventos de sistema e os eventos de aplicação[31].

Os eventos de sistema são desencadeados pelo funcionamento do próprio sistema[32] e da gestão das suas primitivas de base, não podendo ser modificados pela aplicação. Ao criar uma entidade (grupo ou entidade elementar), são configurados um conjunto de eventos base, comuns a todas as entidades do sistema. Existe um conjunto de tipos de eventos que são "subscritos" implicitamente no acto de criação ou de filiação num

[31] As entidades podem publicar eventos para o grupo, desde que façam parte do grupo.

[32] A descrição dos eventos de sistema e os comportamentos que desencadeiam são descritos no capítulo 5, onde é descrita a realização de um protótipo do modelo.

grupo. Os eventos de aplicação são eventos definidos ao nível da aplicação, sendo o seu tratamento, ou seja método de atendimento, definido por esta.

No modelo, é definida uma estrutura de suporte de eventos para cada entidade[33]. Nessa estrutura, são definidos os eventos que são produzidos e os que são processados pela entidade, correspondendo esta estrutura à que suporta a lista de eventos anunciados ("advertised") e subscritos ("subscribed"). Também o representante de cada grupo mantém a informação referente aos eventos que foram anunciados e à identificação dos seus produtores (para os eventos de aplicação). Ao entrar num grupo, todos os membros passam a receber todos os eventos ocorridos dentro desse grupo, podendo cada um optar por proceder ao seu tratamento. Os eventos são disseminados sob a forma de mensagens que são caracterizadas por um identificador de evento e pelo produtor do evento, sendo recebidas por métodos de atendimento existentes nos membros e processados de acordo com o seu tipo.

Ao entrar num grupo, todos os membros passam a receber os eventos destinados ao grupo. No caso de eventos específicos da aplicação, deve ser efectuada a acção de *advertise*, de modo a publicitar para o grupo a existência de um novo tipo de evento que os seus membros podem subscrever. A subscrição corresponde à configuração da estrutura de tipos de eventos, onde é indicado o atendimento que deve ser executado quando da ocorrência do evento.

```
...
adv_type = ACTIVE
{ definição da estrutura de atributos de evento com o formato
   <nome, valor>  criação de evento }

// neste caso é feita a actualização das estruturas de suporte de eventos
// com a informação referente ao novo tipo de evento e
/ desencadeia evento de aviso de novo evento disponível aos membros do grupo
event_type = { atribuiçao de uma identificação ao novo tipo de evento }

advertise(in: my_id, id_group, adv_type, event_type, info)
...

msg = "Olá turma"
{ Constrói event com identificação do tipo de evento + parametros [+ msg] }

validate = publish( in: d_group, event )
...
```

Listagem 4.10: Definição e anúncio de novo tipo de evento

Para poder processar um dado tipo de evento, os membros devem ter anteriormente subscrito este serviço, ou seja, devem ter acesso ao método de tratamento deste evento. Os membros do grupo podem efectuar a consulta de quais os tipos de eventos existentes (que foram "advertised"), sendo esta informação mantida pelo representante

[33]Essas estruturas são detalhadas no capítulo seguinte onde é descrita a arquitectura de suporte ao modelo.

do grupo. A chamada da primitiva *check_adv* retorna a lista de tipos de eventos que obedecem às características especificadas na chamada.

Consulta de tipos de eventos

```
...
id_provider = null                  // de qualquer produtor
ev_type = null                      // todos os eventos

id_group = {identificador do grupo}

[adv_list] = check_adv(in: id_group; out: lista_tipo_evento)

// é retornada a lista_tipo_evento de todos os tipos de eventos
// "advertised" no grupo especificado
// cada elemento da lista contem <ev_type, id_produtor>
// ou seja, tipo de evento e entidade que o produz
...
```

Listagem 4.11: Consulta de tipos de evento existentes num dado grupo

A subscrição de um evento (de um tipo que foi previamente anunciado), é feita através da chamada da primitiva *subscribe*(). Esta primitiva procede à configuração da estrutura de eventos do membro, que passa a efectuar também o processamento desse novo tipo de evento. O processamento que deve ser desencadeado quando da recepção do tipo de evento é descrito na rotina de atendimento que lhe é associada, quando da invocação da primitiva *subscribe*()[34].

Subscrição

```
...
ev_type = { tipo de evento - identificação do tipo de evento }
ev_handler = { rotina de atendimento associada pela entidade ao tipo de evento }

validate = subscribe(in: id_group, ev_type, ev_handler)
...
```

Listagem 4.12: Subscrição de um evento de um dado grupo

4.6.7 Actualização de dados partilhados no grupo

No tipo de cenários de utilização que estiveram na base do desenvolvimento do modelo proposto, as funcionalidades referentes à partilha e acesso a dados dentro do espaço de um grupo assumem especial relevo. No modelo proposto, existe um conjunto

[34]Note-se que todos os eventos são recebidos mas podem nem todos ser processados, ou seja, são enviados aos destinatários, mas nestas pode não ser desencadeado nenhum processo de tratamento. No caso das entidades estarem OFFLINE estas não vão receber os eventos que entretanto tenham ocorrido.

de primitivas de gestão de dados partilhados pelos membros do grupo, para inserção e actualização de dados e também para a sua posterior consulta e acesso.

No exemplo apresentado seguidamente, é retratada uma situação em que um dos membros do grupo efectua a actualização de dois ficheiros que se pretendem tornar acessíveis ao grupo: um deles passa a estar acessível aos restantes membros (que recebem uma notificação de alteração); o outro fica com acesso restrito ao seu produtor, ou seja, os restantes membros do grupo não lhe podem ter acesso.

A primitiva de actualização de dados no espaço partilhado do grupo é a primitiva *update*(). Através desta primitiva, é criado um tuplo com a informação referente (de acordo com critérios definidos pela aplicação) ao ficheiro e, de um modo geral, a forma de lhe aceder ("caminho" para o sistema onde os ficheiros são armazenados). A semântica de caracterização dos dados, ou seja o conjunto de *tags*[35] que caracterizam o conteúdo dos dados, é deixada à responsabilidade da aplicação, assim como a sua gestão e manutenção.

A primitiva *update* envolve duas acções distintas:

- Criação do tuplo, com base nos argumento de invocação da primitiva;
- Inserção no espaço partilhado do grupo, do tuplo referente aos dados[36].

Update 1: Actualização, com notificação dos membros do grupo de dados novos para consulta

```
...
// parâmetros gerais
tipo_acesso = PUBLIC
tipo_notificação = NOTIFY_UPD

// parâmetros específicos da aplicação, por exemplo:
tipo_ficheiro = jpeg
comentario = "torre belem"
[tags] = [exterior, pessoas, monumento, histórico, arquitectura]
ref_dados = { acesso aos dados no SI }
data = 19/04/2006

// construção da lista de dados com base nos parâmetros da aplicação
dados = { construção dos dados com base nos parâmetros específicos definidos pela
    aplicação }

valide = update(in: produtor_id, grupo_id, tipo_acesso, tipo_notificação, dados )

if ( valide == OK )
        // updade efectuado com sucesso
else
        // update inválido, deve repetir operação
...
```

[35] As *tags* disponíveis para a caracterização dos conteúdos são definidas pela aplicação.

[36] É da responsabilidade da aplicação a construção da lista de dados, podendo um dos campos ser uma referência para o acesso aos dados no sistema de informação.

Listagem 4.13: Colocação de dados para partilha num grupo

No exemplo descrito, foi efectuada uma actualização com notificação, ou seja a primitiva *update*() difunde pelos membros do grupo a informação referente à acção de actualização de dados no espaço partilhado. Os membros recebem um evento de actualização com a informação referente aos dados alterados e do seu produtor, sendo da responsabilidade da aplicação definir qual o comportamento a ser desencadeado. Pode-se optar por efectuar uma leitura e visualização ou proceder à actualização da sua lista de dados partilhados no contexto do grupo especificado. Este tipo de evento (*update* de dados) faz parte dos eventos de sistema, ou seja, ao filiar-se num grupo, todos os membros o subscrevem implicitamente.

Update 2: Actualização privada de dados

```
...
// parâmetros gerais
tipo_acesso = PRIVATE
tipo_notificação = NO_NOTIFY_UPD

// parâmetros para construção da lista de dados,
// com base no definido pela aplicação
{ parâmetros referentes aos dados }

dados = { construção dos dados com base parâmetros de dados da aplicação}

validate = update(in: produtor_id, grupo_id, tipo_acesso, tipo_notificação, dados )

if ( validate == OK )
        // updade efectuado com sucesso
else
        // update inválido, deve repetir operação
...
```

Listagem 4.14: Colocação de dados próprios no espaço do grupo

Neste segundo exemplo, a informação só pode ser consultada e acedida pelo produtor da informação, que poderá, no entanto, vir a alterar o seu tipo de acesso, da seguinte forma:

```
...
tipo_acesso = PUBLIC
tipo_notificação = NO_NOTIFY_UPD

// parâmetros para construção da lista de dados,
// com base no definido pela aplicação
{ parâmetros referentes aos dados }

dados = { construção dos dados com base parâmetros de dados da aplicação}

// acesso aos dados aberto a todos os membros do grupo
```

```
set_access(in: my_id, grupo_id, tipo_acesso, dados )
...
```

Listagem 4.15: Alteração dos atributos dos dados colocados no espaço partilhado

4.6.8 Consulta de informação no espaço partilhado

Como já foi referido anteriormente, ao colocar a informação no espaço partilhado, é armazenado um conjunto de características, definidas pela aplicação, referentes aos dados (por exemplo: *tags*, data, produtor da informação, etc.). Essa informação, que faz parte do tuplo definido para os dados, permite aos membros do grupo efectuarem pesquisas personalizadas dos dados colocados no espaço, através da primitiva $find()$. Uma forma de utilizar essa primitiva é apresentada de seguida, onde é efectuada uma pesquisa com base em especificações indicadas através de palavras chave, sendo depois consultados e visualizados os dados.

```
...
produtor_id = null              // todos os produtores de dados (membros do grupo) são
     considerados
sinc = NO_BLOCK // pesquisa com semântica não bloqueante

// parâmetros específicos da aplicação, por exemplo:
tipo_ficheiro = jpeg
comentario = -
[tags] = [exterior, pessoas, -, -, -]
ref_dados = { acesso aos dados no SI }
data = 19/04/2006

// construção da lista de dados com base nos parâmetros da aplicação
dados_pesquisa = { construção dos dados com base nos parâmetros específicos definidos
    pela aplicação }

find( in: my_id, grupo_id, produtor_id, sinc, dados_pesquisa; out: num )
// se num = 0 não foi encontrado nenhum tuplo no espaço
// que satisfaça as condições

for ( i=0 ; i <num ; i++ )
{
        // consulta dados na estrutura interna que suporta
        // a pesquisa de tuplos do espaço
        consult(in: i; out: tuplo_dados )

        dados = { a partir do tuplo_dados extrai os dados necessários à aplicação }
        ...
}
...
```

Listagem 4.16: Consulta de dados do espaço partilhado de um dado grupo

No caso dos dados conterem somente a referência de acesso no sistema de informação (SI), a consulta dos mesmos pode não ter sucesso, já que os dados podem ter sido entretanto removidos por outro membro (entre a chamada da primitiva $consult()$

e a visualização ao nível da interface de utilizador) ou pode ocorrer um problema de ligação com o SI. Não é garantido que, entre a chamada de $find()$ e $consult()$, não tenha ocorrido uma remoção ou outra alteração dos dados.

Outra forma de consulta de dados no espaço partilhado do grupo, pode ser efectuada sem recurso prévio à primitiva $find()$, como é apresentado no exemplo seguinte de utilização.

```
...
produtor_id = null                        // todos os produtores de dados (membros do
    grupo) são considerados
sinc = NO_BLOCK          // pesquisa com semântica não bloqueante

// parâmetros específicos da aplicação, por exemplo:
tipo_ficheiro = jpeg
comentario = -
[tags] = [exterior, pessoas, -, -, -]
ref_dados = { acesso aos dados no SI }
data = 19/04/2006

// construção da lista de dados com base nos parâmetros da aplicação
dados_pesquisa = { construção dos dados com base nos parâmetros específicos definidos
    pela aplicação }

consult( in: my_id, grupo_id, produtor_id, sinc, dados_pesquisa; out: tuplo_dados )

// no caso de existir tuplo que satisfaça as condições, e retornado
// se existir mais do que um, é retornado um de entre os que
// satisfazem as condições
// se não existir retorna tuplo vazio
...
```

Listagem 4.17: Pesquisa de dados no espaço partilhado de um dado grupo

Ao longo desta secção, foram descritos alguns exemplos isolados de utilização das primitivas disponibilizadas pelo modelo. Na secção seguinte, são descritos dois cenários globais, onde o modelo é utilizado como forma de definir e caracterizar as interacções entre os participantes e a sua estruturação em grupos organizados de interesse.

4.7 Cenários de aplicação

O modelo aqui proposto define um conjunto de primitivas, concebidas com o objectivo de facilitar a programação de aplicações que promovam a interacção entre utilizadores num determinado espaço. No ambiente computacional suportado pelo modelo, as aplicações requerem um conjunto de funcionalidades e serviços que podem ser, de modo simples, modelados com recurso às primitivas propostas pelo modelo.

Analisando alguns dos cenários possíveis de aplicação, foram modeladas algumas das funcionalidades e serviços, que permitem ilustrar a forma de utilização do modelo, com especial ênfase na formação e gestão de grupos e nos diferentes mecanismos de comunicação suportados.

Diversas semânticas de difusão de mensagens e de gestão de membros são suportadas, podendo ser configuradas de acordo com as especificações da aplicação em causa.

Na maioria das aplicações típicas analisadas, observou-se que, de um modo geral, se privilegia a simplicidade e flexibilidade e não a complexidade dos mecanismos envolvidos. Os exemplos propostos tentam captar as necessidades básicas de alguns dos serviços criados e que foram desenvolvidos com base no modelo proposto, que captam a dinâmica de grupos, os mecanismos de interacção entre os membros, assim como alguns aspectos relacionados com mobilidade. Neste capítulo e posteriormente no capítulo onde são descritas aplicações, analisa-se até que ponto o modelo é útil, transparente e flexível para o desenvolvimento de aplicações.

De seguida, é feita uma breve descrição dos dois cenários de aplicação, colocando em evidência algumas das funcionalidades desenvolvidas, tendo por base o modelo proposto. O primeiro exemplo refere-se a um *campus* universitário e a possíveis situações que podem ocorrer dentro desse espaço. O segundo exemplo ilustra situações que podem surgir num aeroporto e que podem ser modeladas com o sistema proposto. Tratam-se de dois ambientes diferentes, que promovem diferentes tipos de interacções entre utilizadores. No entanto, os tipos de serviços são semelhantes, embora no primeiro cenário seja dado um maior foco aos grupos explícitos e no segundo, aos grupos implícitos.

4.7.1 *Campus* universitário

Num ambiente como um *campus* universitário, existe um grande número de utilizadores, possuidores de dispositivos móveis (como portáteis e PDAs), existindo, para além disso, uma predisposição por parte dos utilizadores, para a formação de grupos de contactos. Essa predisposição deve-se, não só às necessidades de trabalho, mas também às exigências de segurança ao nível dos intervenientes nos grupos, que pode não ser garantida em ambientes mais "abertos". A geração e manutenção de grupos é relativamente estável[37] e baseada, de um modo geral, nas relações e necessidades profissionais dos seus participantes. Obviamente, também podem surgir diversos grupos gerados a partir dos dados pessoais dos utilizadores, com base nas suas características e interesses.

Um exemplo de tipos de grupos que podem ser observados neste cenário de utilização é esquematizado na figura 4.8:

- grupo de professores de informática (Profs. Informática)
- grupo de trabalho criado pelo aluno João (Grupo João)
- equipa de futebol (FTeam)

[37] Por exemplo, com períodos de actividade na ordem do semestre, ano ou múltiplos anos, correspondendo a acções de formação ou de investigação.

- grupos associados a turmas/disciplinas (Programação (P1), Matemática I)
- grupo de utilizadores da sala 123

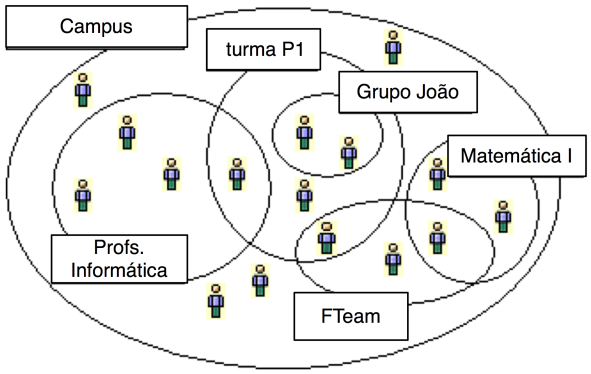

Figura 4.8: Exemplos de grupos num cenário de um *campus* universitário

Os diversos grupos criados podem ter diversos tipos de utilização. No entanto e de um modo geral, tendem a ser utilizados como forma de difundir e partilhar informação, como por exemplo: a alteração da data de um exame (difundido para o grupo dos alunos da disciplina); a marcação de uma reunião de trabalho; a partilha de um conjunto de slides, vídeo e anotações das actividades de uma turma.

A criação e alteração da constituição dos grupos podem obedecer a diferentes formas de caracterização, por exemplo:

- uma lista de participantes pré-definida quando da criação do grupo, em que somente os elementos que constam da lista se podem tornar membros desse grupo (grupo associado a uma disciplina ou turma, por exemplo); a entrada de novos membros tem que ser aprovada pela maioria dos participantes do grupo;
- a entrada de um membro tem que ser validada por, pelo menos, um membro ou por um membro específico do grupo (o seu criador, por exemplo, ao qual podem ser atribuídos privilégios especiais);
- todos os utilizadores do sistema, que obedeçam a um determinado grupo de características, passam implicitamente a ser membros de um dado grupo (criado com base em parâmetros configuráveis).

Umas das necessidades das aplicações neste cenário é a existência de um mecanismo que defina o estado de um membro no grupo. Após a sua entrada no grupo (e por definição do modelo), o novo membro passa a fazer parte da lista de membros aceites no grupo, sendo necessário, de um modo geral, definir explicitamente a sua saída. No entanto, poderá alterar-se o seu estado dentro do grupo, procedendo explicitamente à alteração do seu estado de actividade[38], como já foi anteriormente referido.

[38]O estado da actividade pode representar o facto da entidade se encontrar, por exemplo: desligada, bloqueada ou inacessível.

Os grupos podem também ser utilizados ao nível da aplicação como forma de trocar informação, como por exemplo colocar membros em contacto entre si, de acordo com os seus interesses mútuos. Por exemplo, a informação associada ao grupo de utilizadores que estão interessados em partilhar uma casa, pode ser cruzada com a do grupo de utilizadores que pretendem alugar um quarto e os respectivos membros serem assim, notificados deste tipo de informação. Os grupos implícitos podem ser utilizados para modelar este tipo de situações de forma simples, de acordo com o especificado por um conjunto (conjunção) de regras que representem as condições que caracterizam o grupo.

De seguida, são apresentadas, resumidamente, algumas das situações de utilização neste cenário e as sequências de acções que devem ser desencadeadas para cada passo.

Criação e filiação num grupo

Um aluno, que se encontra inscrito no sistema, pretende criar um grupo onde ele e os seus parceiros de estudo possam partilhar informação e comunicar entre si. O primeiro membro deste grupo é o seu próprio criador ficando, a partir daí, a entrada de novos membros sujeita à aprovação pela maioria dos membros do grupo. Para criar este grupo, deverão ser efectuados os seguintes passos:

1. definir as configurações do grupo necessárias para a aplicação (política de filiação, dimensão e tempo de existência do grupo, lista de serviços do sistema a que deve ter acesso);

2. invocar a primitiva de criação (*create*()), passando a informação referente às características do grupo e identificação do criador;

3. invocar a primitiva de filiação (*join*()) no grupo criado.

Como já foi referido, após a filiação do primeiro membro (o criador), os restantes são submetidos a um processo de votação para entrada.

A primitiva de filiação do modelo tem diferentes políticas pré-definidas, sendo a sua configuração (selecção) efectuada quando da criação do grupo. Algumas das políticas são: admissão directa; aprovação por parte do criador do grupo; aprovação de um dos membros; aprovação da maioria dos membros. No caso de ser criada uma lista de admissão, essa informação é passada quando da criação do grupo, especificando que se trata da um política de admissão baseada numa lista de "convidados".

Envio de mensagem

Outra situação possível é a de um professor que pretende enviar uma mensagem, para o conjunto de alunos da sua turma, indicando a alteração de uma data de exame. Ao

tornarem-se parte de um grupo, os membros passam a ter acesso a diversos mecanismos de comunicação, que podem utilizar de acordo com as suas necessidades. Neste caso, para o envio de uma mensagem, a aplicação pode optar por duas soluções distintas, utilizar a primitiva *send*() ou utilizar a comunicação por eventos:

- ao enviar uma mensagem através da primitiva *send*(), a aplicação está a determinar que o envio seja atendido do lado dos receptores, através da primitiva de recepção de mensagens *receive*(), tendo de ser desencadeada uma acção explícita de recepção por parte do destinatário;
- através do mecanismo de eventos, o envio de uma mensagem desencadeia um evento que será atendido por um método de atendimento do lado do receptor, que interrompe o seu processamento normal para atender a recepção da mensagem, sendo este método invocada de forma implícita[39].

Difusão de informação

Na situação em que um aluno, pertencente a um grupo de estudo, pretende avisar os seus colegas (membros do grupo) que se encontra a estudar na biblioteca, pode optar por publicar essa informação (tipo de evento), sendo notificados os membros do grupo que se encontram activos. Neste caso, esta situação pode ser modelada através do mecanismo de eventos, que permite a difusão de informação entre os membros de num grupo.

O mecanismo de eventos é utilizado sempre que se pretende efectuar a difusão de informação para todos os membros dentro do espaço de um grupo, por exemplo do caso de envio de mensagem em que o receptor é um grupo, a política de recepção implementada por este pode desencadear a difusão.

Partilha de dados

Nesta situação, o professor pretende, por exemplo, colocar disponível um vídeo para ser acessível por toda a turma. A sequência de acções para atingir os objectivos desta tarefa é a seguinte:

1. colocar o ficheiro de vídeo no sistema de informação;
2. colocar a referência e os dados relativos ao vídeo (e que dão acesso ao ficheiro), no espaço partilhado do grupo "turma 2A".

Os membros notificados podem aceder aos dados, consultando o espaço partilhado para obter a referência de acesso ao ficheiro de vídeo no sistema de informação.

A primitiva do modelo que actualiza dados para partilha dentro de grupo é a primitiva *update*, que executa a sequência de acções descrita.

[39]Note-se que o tratamento de eventos do lado do receptor pode recorrer a outro tipo de mecanismo de execução concorrente por exemplo baseado na activação de *threads*.

Consulta de dados

Para consultar os dados disponíveis no espaço partilhado do grupo, é efectuada uma consulta ao espaço de memória partilhada, de acordo com o critério de pesquisa pretendido, sendo retornadas as referências referentes aos dados que obedecem a esses critérios. Por exemplo, no caso de um aluno pretender consultar os *slides* da apresentação do trabalho de um dos seus colegas, deve definir o critério de pesquisa, neste caso especificando o tipo de dados a consultar e o seu produtor. Ser-lhe-á retornado o conjunto de dados que obedecem a essas especificações, sendo opção dele seleccionar o pretendido que será transferido do sistema de informação. Note-se que não é garantida a consistência dos dados, uma vez que entre a consulta dos dados no espaço partilhado e o acesso ao sistema de informação, o produtor dos dados poderá ter removido o ficheiro de dados do sistema.

4.7.2 Aeroporto

Outro cenário possível de aplicação considerado é o espaço de um aeroporto. Trata-se de um espaço com utilizadores bastantes distintos do descrito anteriormente, com necessidades e características de utilização diferentes. No entanto, ao nível das aplicações, podemos observar que existem padrões de interacção idênticos.

Neste cenário, existe uma muito maior heterogeneidade de utilizadores (e de dispositivos), enfrentando este espaço diferentes problemas, ao nível de acesso e confiança por parte dos utilizadores. Uma aplicação, para este tipo de ambiente, deverá ter em conta diferentes aspectos, como por exemplo: a convergência de interesses por parte dos utilizadores (maior heterogeneidade de utilizadores); o tempo de estadia no espaço (maior volatilidade de utilizadores); o número de pessoas "desconhecidas" é bastante maior; a origem dos grupos deve ser de um modo geral definida pelo sistema (grupos implícitos), de acordo com o tipo e quantidade de utilizadores que existam no espaço do aeroporto; existe uma maior predisposição por parte dos utilizadores para aplicações do tipo informativo do que do tipo colaborativo.

Neste cenário de utilização, assume especial importância o suporte implícito de serviços de apoio ao público. Pode-se assumir que um dos objectivos da entidade gestora do aeroporto é disponibilizar informação útil e interessante aos viajante. Para tal pode fomentar a utilização, por parte dos viajantes, dos canais de comunicação existentes, de modo não só a fornecer informação útil, mas também promover a cooperação entre viajantes possuidores de interesses comuns assim como divulgar e optimizar a utilização dos serviços disponíveis no aeroporto. Casos simples de utilização podem ser por exemplo, a partilha de um taxi, a descoberta de um meio de transporte específico, o encontro não planeado de amigos ou a simples descoberta de indivíduos com a mesma nacionalidade ou pertencendo a um mesmo grupo social ou profissional. Por exemplo, um passageiro que chega ao aeroporto inserido num grupo ou que pretenda reunir-se

ao seu grupo terá ao seu dispor a possibilidade de convocar explicitamente um canal de comunicação com o grupo. Explicitamente, o passageiro declara a sua intenção de entrar em contacto com os membros do grupo e pode configurar localmente a entidade que coordena a sua interacção com o sistema, de modo a receber uma notificação sobre a chegada dos restantes elementos. O grupo pode manter-se activo até que todos os seus elementos se tenham retirado e ninguém tenha efectuado um pedido no sentido de sustentar, durante mais algum tempo, a informação associada ao grupo.

A informação que cada elemento disponibiliza sobre si é parametrizável de forma a que os restantes membros do grupo possam ter acesso a informação sobre a sua localização, notas, imagens ou ainda outros dados que queiram partilhar. De acordo com a configuração do grupo, este pode manter-se activo, mesmo que os elementos que o formam se encontrem ausentes: desta forma, membros que cheguem *a posteriori* ao aeroporto podem ter acesso ao grupo assim como a toda a informação disponível sobre ele. Esta situação revela o interesse da manutenção temporária da informação referente ao grupo, mesmo quando todos os membros do grupos se encontrem inactivos (também se pode passar informação sobre o grupo ao sistema de informação, de modo a torná-la persistente).

Dinâmica de utilização

Nesta secção, são descritos alguns exemplos da forma como os elementos podem interagir e evoluir dentro do sistema, sendo identificadas algumas das possíveis situações de utilização e a forma como estas são tratadas com o modelo proposto.

Num aeroporto, podem ser identificadas basicamente duas classes distintas de utilizadores, com base no tipo de interacção dentro deste espaço:

- passageiros (estão nesta categoria, não só “viajantes” mas também os acompanhantes), entidades que permanecem durante um intervalo de tempo limitado no espaço;
- trabalhadores do aeroporto (pessoal de terra, operadores turísticos, manutenção, segurança, tripulações, etc ...), "clientes" habituais do espaço.

Nos exemplos aqui apresentados, é dado um maior relevo às situações envolvendo os passageiros, dado que as situações dos trabalhadores são de certa forma similares (da mesma classe) às do exemplo referido anteriormente.

Passageiros: os passageiros são os elementos que existem em maior número num aeroporto, são também os elementos que se encontram activos durante um intervalo de tempo[40] mais curto, e que possuem objectivos bastante diversificados (de acordo

[40]Este é, tipicamente, o tempo de duração do trânsito de um passageiro pelo aeroporto, seja na chegada, partida ou na transferência entre voos.

com as suas características). No entanto, são os que se podem mostrar mais disponíveis para utilizar este tipo de sistema (de modo, por exemplo, a ocupar os tempos de espera), sendo por esse motivo os intervenientes do sistema aos quais foi dada uma maior atenção neste exemplo.

Uma aplicação desenvolvida para ser utilizada por passageiros poderá incluir as seguintes funcionalidades:

- registo no sistema, sendo esta a primeira acção que deve ser desencadeada; somente após o registo, um passageiro passa a ser uma entidade do sistema;
- ao tornar-se uma entidade do sistema, é desencadeado o mecanismo de filiação (automática) nos grupos implícitos criados;
- pode filiar-se explicitamente em grupos já existentes, passando a poder interagir com os outros membros;
- criar novos grupos e destruir grupos que entretanto tenha criado;
- partilhar informação (dados pessoais e conteúdos) com membros dos seus grupos;
- trocar mensagens com membros dos seus grupos ou outras entidades registadas do sistema.

Seguidamente, são apresentados exemplos de situações que ilustram possíveis perfis de utilização recorrendo a grupos explícitos (primeiro exemplo) e a grupos implícitos (segundo exemplo).

Utilização de grupos explícitos: No primeiro perfil de utilização aqui descrito, partiu-se de uma situação bastante habitual num aeroporto.

Uma empresa vai buscar passageiros que chegam ao aeroporto, vindos de dois voos distintos (V1 e V2), para os transportar para o hotel:

- o condutor da empresa de transporte, chega ao aeroporto e regista-se (ou coloca-se *online*) no espaço do aeroporto e cria um novo grupo com a referência da empresa/situação e filia-se nele. A partir desse instante o grupo passa a surgir no espaço de grupos existentes no aeroporto. O condutor fica a aguardar mensagens com pedidos de filiação;
- os passageiros do voo V1 chegam ao aeroporto e sinalizam a sua presença no aeroporto (registo no sistema). Consultam a lista de grupos existentes e tentam filiar-se no grupo que pensam ser o seu. A filiação só é efectuada se a admissão ao grupo for aceite pelo condutor (criador do grupo);

- o condutor envia uma mensagem para o grupo, com informação relativa ao local/hora de encontro;

- o voo V2 sofreu entretanto, um pequeno atraso, ou seja, ter-se-á que aguardar mais um pouco pela chegada dos restantes passageiros. O condutor envia uma mensagem para o grupo (publica aviso) a informar do atraso e que os passageiros podem ir ao bar/lojas, garantindo que, quando os restantes elementos chegarem, serão notificados;

- os passageiros circulam pelo aeroporto e aguardam notificação por parte do condutor;

- os passageiros do voo V2 chegam e efectuam o seu registo no sistema e filiam-se no grupo (admissão é validada pelo condutor);

- o condutor envia uma mensagem para o grupo a notificar local/hora de partida;

- entretanto um dos passageiros do voo V2 perdeu a bagagem e notifica o grupo de que irá ficar retido durante mais tempo no aeroporto;

- o condutor envia uma mensagem directa para o passageiro, a informar que irá levar os restantes passageiros ao hotel e que irá voltar mais tarde;

- o condutor coloca-se num estado de desligado (temporariamente no grupo e no aeroporto);

- o condutor volta ao aeroporto (coloca-se *online*) e envia mensagem para o grupo (que agora só tem um passageiro), a avisar que está de volta e do local onde se encontra;

- ao sair novamente do aeroporto, o condutor elimina finalmente o grupo.

Utilização de grupos implícitos: a entidade responsável pela gestão do aeroporto tem acesso à aplicação que lhe permite criar e configurar grupos implícitos. Estes podem ser criados com base nas preferências e características preenchidas pelos passageiros (entidades elementares do sistema)[41]

Foram definidos diversos conjuntos de características que um administrador considerou relevantes e de interesse para a criação de grupos. Um exemplo de um dos grupos criado, tomou como base as características da nacionalidade e motivo da viagem instanciadas com os valores, por exemplo, "espanhola" e "turismo" respectivamente. É

[41] As características e preferências fazem parte dos dados específicos da aplicação/problema que se pretende modelar e foram por esta definidas.

então formado o grupo de turistas espanhóis que inclui todos os passageiros registados, que possuam simultaneamente definidas, com estes valores, estas duas características[42]. Este tipo de grupo, assim criado, pode ser utilizado como meio de divulgação de informação mais especializada e personalizada, para o tipo de passageiros que dele fazem parte. Neste caso, por exemplo, poderia ser utilizado pelo administrador para divulgar informação, em língua espanhola, referente à meteorologia ou a empresas que promovem excursões pela cidade. Os membros do grupo podem também divulgar informação para o grupo, como é descrito no seguinte exemplo:

- um passageiro pertencente a este grupo e que, entretanto, verificou que o seu voo sofreu um atraso significativo, decidiu fazer uma pequena excursão pela cidade para visitar um ou dois locais turísticos. Como gostava de ter companhia para partilhar um táxi, decide criar um grupo de excursão com número máximo de quatro elementos e envia uma mensagem para o grupo a informar deste facto, questionando se existem interessados em participar;

- os interessados filiam-se no grupo, até ao valor máximo permitido pelo grupo. O responsável pela iniciativa, ao verificar que o grupo excursão, que tinha criado, possui novos membros, emite duas mensagens: uma, para o grupo de turistas espanhóis, a informar que a excursão está preenchida; e outra, para o grupo "excursão", a avisar do local de encontro. Disponibiliza também para o grupo, um ficheiro com a sua foto, para que possa ser mais facilmente identificado;

- este grupo "excursão", pode ainda ser utilizado para proceder à troca de fotos e vídeos (referentes à visita) entre os participantes.

4.8 Conclusão

Ao longo deste capítulo foi descrita a proposta de um modelo de grupos para aplicações interactivas distribuídas (MAGO) bem como o conjunto de primitivas por ele suportado.

O conjunto de primitivas desenvolvido permite descrever, de forma simples, os possíveis padrões de utilização no desenvolvimento de aplicações interactivas distribuídas, em particular na definição e criação de funcionalidades associadas às estruturas de grupos.

No final do capítulo, deu-se uma visão geral da forma como o modelo pode ser utilizado em cenários reais de aplicação. Existem muitos outros exemplos de cenários de aplicação que poderiam ter sido descritos, por exemplo em espaços como museus, centros comerciais e centros de congressos. Tratam-se de espaços onde, de um modo

[42]Ou que entretanto se registem e satisfaçam essas condições.

geral, os utilizadores tendem a formar grupos de interesse (explícitos ou implícitos) e a interagirem entre si dentro destas estruturas.

Mais à frente, no capítulo 6, é analisada a forma de desenvolver e implementar aplicações suportadas pelo protótipo do modelo desenvolvido, sendo a arquitectura e realização do protótipo de suporte do modelo, descritas ao longo do próximo capítulo.

5

Arquitectura de suporte ao modelo

Conteúdo

Neste capítulo é descrita a arquitectura que suporta o modelo proposto no capítulo 4. É feita uma apresentação da plataforma sobre a qual foram desenvolvidas as primitivas do modelo e são analisadas as questões relacionadas com a realização das mesmas. São também apresentados os elementos da arquitectura que suportam a interacção com o sistema de informação e a gestão de grupos implícitos.

5.1 Introdução

Ao longo desta secção analizam-se as infra-estruturas e plataformas que dão suporte ao modelo proposto, sendo discutidas as opções tomadas.

O modelo proposto situa-se entre as aplicações e uma camada de suporte da plataforma de computação (figura 5.1). Um programador de aplicações tem ao seu dispor, através deste modelo, um conjunto de mecanismos básicos que pode invocar através da chamada directa das primitivas e de serviços básicos que permitem configurar e desenvolver aplicações como as ilustradas nos cenários considerados como casos de estudo.

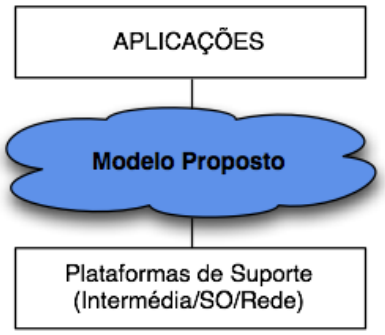

Figura 5.1: Integração do modelo proposto

Para a realização do modelo, propõe-se a seguinte organização da arquitectura do sistema, ilustrada na figura 5.2, na qual se representa a forma como os diversos elementos constituintes se situam ao nível de camadas da arquitectura, observando-se que, na sua base, se situam as camadas de rede e de sistema de operação. Estas camadas inferiores oferecem suporte de comunicações e de operação.

Sobre a camada de sistema de operação e de rede, situa-se uma camada intermédia (*middleware*) de suporte às formas de comunicação de grupo, por eventos e por espaços partilhados. É esta camada que oferece os mecanismos de base necessários para a realização das interacções definidas pelo modelo proposto.

O desenvolvimento de aplicações é feito sobre a plataforma de modelo, tirando partido das suas primitivas e funcionalidades, em particular os mecanismos que gerem as interacções entre entidades e a sua estrutura organizacional. Articulado com a arquitectura, existe um sistema de informação que dá suporte à gestão de dados da aplicação e que também serve de suporte ao modelo.

Observando de forma mais detalhada os módulos constituintes da arquitectura (figura 5.2), podem ser identificados os elementos responsáveis pela realização das diversas classes de primitivas descritas no modelo (capítulo 4).

Ao nível dos mecanismos básicos oferecidos pelo modelo, identificam-se:

- o módulo de comunicações, que agrega todo o conjunto de funcionalidades asso-

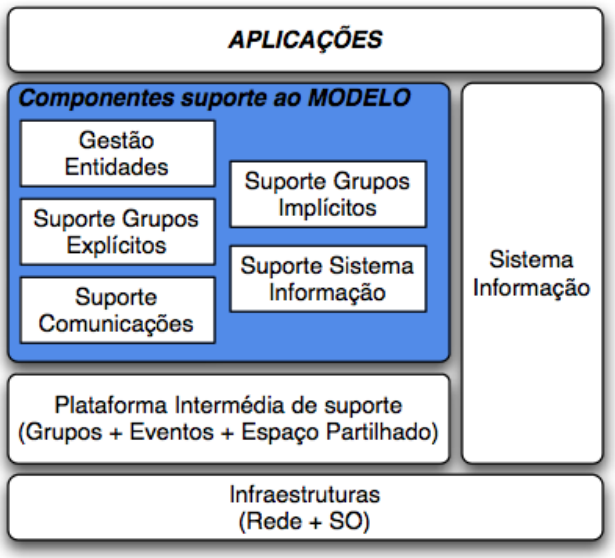

Figura 5.2: Níveis e módulos constituintes da arquitectura proposta

ciadas às comunicações, sendo estas utilizadas, não só ao nível da aplicação, mas também como suporte para a implementação das funcionalidades dos restantes módulos;

- o módulo de suporte à gestão da dinâmica de grupos explícitos, no qual estão reunidas as funcionalidades associadas às interacções de elementos enquanto membros de grupos;

- o módulo de gestão de entidades, responsável pela gestão das entidades do sistema, enquanto elementos que possuem identificação e atributos próprios, que os identificam e caracterizam enquanto membros do sistema;

- o módulo de suporte à interacção com o sistema de informação, no qual estão definidas as funcionalidades que permitem a gestão e acesso dos dados referentes ao modelo, sendo possível também às aplicações recorrerem directamente às suas funcionalidades como forma de acesso ao sistema de informação;

- o módulo de suporte à gestão dos grupos implícitos, que se encontra directamente ligado ao módulo de suporte ao sistema de informação e é responsável pela criação e gestão particular destes grupos.

Ao longo desta secção discutem-se, de forma mais detalhada, as camadas de suporte presentes na arquitectura proposta.

5.2 Infra-estrutura de rede e dispositivos computacionais

Na base da arquitectura (figura 5.2), tem-se a camada de suporte de rede, sistema de operação e de ambiente de execução. A este nível, parte-se do pressuposto de que existe uma infraestrutura física de suporte às comunicações, assente nos protocolos TCP/IP, sendo essa rede construída com base em ligações fixas estacionárias e em ligações sem fio. No caso das ligações sem fio, assume-se que estas têm por base o protocolo Wi-Fi (802.11g) [Gas05], sendo esta a infraestrutura que suporta as ligações dos dispositivos móveis considerados[1], com o sistema de suporte aos serviços.

A rede fixa estacionária é essencialmente constituída pelas máquinas onde se encontram os servidores e os mecanismos de gestão central do sistema (como por exemplo o sistema de informação), que servem de base aos diversos serviços utilizados como suporte e/ou directamente pela aplicação.

Ao nível do sistema de operação, o requisito de base necessário é a existência de uma plataforma de execução de programas Java (Java Runtime) [vdL02, GPB+06] com os mecanismos associados a invocação remota de métodos (RMI - *Remote Method Invocation* [McC97]), uma vez que estes são utilizados pela plataforma intermédia[2] e pelos servidores, para a gestão dos processos das comunicações.

Os dispositivos computacionais do sistema foram classificados em duas classes distintas:

- computadores fixos, com maior poder computacional relativamente aos dispositivos móveis e com maiores garantias de fiabilidade também, sendo nestes que são executados os processos servidores;

- dispositivos móveis, tipicamente com menor capacidade computacional do que os anteriores, com ligações através da rede sem fio e menores garantias de fiabilidade, sendo este o equipamento de acesso tipicamente utilizado pelos clientes das aplicações.

São, obviamente, os dispositivos móveis, aqueles que maiores restrições e problemas colocam ao desenvolvimento de aplicações, devido às suas diferentes características e heterogeneidade, dada a existência de uma grande diversidade dos tipos de dispositivos.

No desenvolvimento de aplicações, nos ambientes como os que foram anteriormente descritos, é de pressupor que o acesso dos utilizadores seja efectuado através de um conjunto heterogéneo de dispositivos móveis. Depende obviamente da aplicação, a criação de um conjunto de serviços adaptados aos diferentes dispositivos, sendo este um factor importante a considerar na realização, devido aos constrangimentos

[1] Assume-se que dispositivos móveis, tais como computadores portáteis, PDAs (assistentes digitais pessoais) e telemóveis, dão suporte a este protocolo de comunicação.

[2] Mais precisamente pela plataforma escolhida JGroupSpace, que se analisa na secção 5.3.

naturais colocados pela utilização destes dispositivos. Os problemas que surgem no desenvolvimento de aplicações colocam-se, não só ao nível da menor capacidade computacional e de armazenamento, mas também ao nível das interfaces e facilidade de utilização, devido às suas dimensões e características de interacção com o utilizador, quando comparados com computadores fixos. Nestes casos, as soluções existentes passam por distribuir o processamento e o armazenamento dos dados entre o "pequeno" dispositivo móvel e um servidor remoto, que funciona como uma extensão virtual do dispositivo. Esta virtualização é realizada ao nível da plataforma intermédia e infraestrutura de rede e sistema de operação. Deste modo, qualquer dispositivo pode executar uma versão simplificada da aplicação, podendo manter remotamente o armazenamento dos dados[3] e eventualmente algum tipo de processamento que necessite de maior capacidade computacional.

5.3 Plataforma intermédia - JGroupSpace

Após uma análise dos diferentes mecanismos de interacção do modelo, observou-se que cada um deles possui efectivamente características e funcionalidades desejáveis para criar uma plataforma com as infraestruturas necessárias ao desenvolvimento de uma implementação do modelo MAGO. A opção tomada foi a de escolher uma plataforma que tornasse disponível uma camada intermédia que cumprisse o maior número de requisitos de base do modelo. A escolha recaiu no sistema JGroupSpaces, um sistema desenvolvido no grupo [Cus08].

O JGroupSpace, que assenta no modelo GroupLog, tem disponível um conjunto de primitivas para a gestão dinâmica de grupos e para a interacção baseadas em troca de mensagens, espaço partilhado de tuplos e eventos (5.3). A concepção das funcionalidades a integrar no JGroupSpace foi, ela própria, influenciada pelos requisitos da definição do modelo MAGO, em particular no que se refere às funcionalidades de suporte a eventos e a espaço de tuplos.

No sistema JGroupSpace um processo é uma entidade activa que executa um programa Java e que, tendo acesso a uma biblioteca oferecida pelo sistema, pode integrar-se em grupos de processos [Cus08].

As funcionalidades estão disponibilizadas através de uma interface de programação (API) em Java e estão separadas em quatro categorias principais:

- **Acesso ao grupo**: os grupos são criados de forma explícita quando um processo invoca a primitiva de filiação, os grupos são identificados por um identificador global. As primitivas que permitem a entrada, saída e obtenção da constituição de um grupo são:

[3]Principalmente as aplicações com maiores exigências computacionais e maiores volumes de dados.

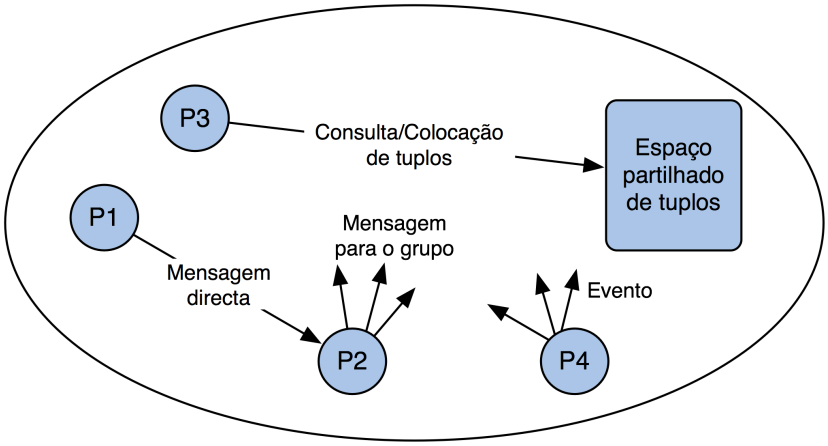

Figura 5.3: Visão de um grupo no JGroupSpace [Cus08]

  - `JGSAddress join(String group_name)`
    Junta um processo a um grupo identificado por *group_name*; se o grupo ainda não existir é criado quando da filiação do primeiro membro;
  - `void leave()`
    O processo invocador sai do grupo;
  - `JGSMembership getMembership()`
    Depois de um processo se juntar a um grupo, é possível obter informação sobre a constituição deste, ou seja dos membros que actualmente se encontram filiados.

- **Envio e recepção de mensagens**: os processos presentes num grupo podem comunicar directamente com os outros membros através de mensagens. As primitivas para a comunicação por mensagens são:

  - `void send(JGSMessage msg)`
    Envia uma determinada mensagem no contexto do grupo. Na mensagem, são especificados, a mensagem, os endereços do emissor e do receptor, note-se que, caso não seja especificado o receptor, a mensagem é enviada para todo o grupo;
  - `JGSMessage receive(int timeout)`
    Recebe uma mensagem proveniente do grupo. Opcionalmente pode ser indicado um *timeout*, especificando o período de tempo que o processo deve aguardar pela recepção da mensagem. Caso esse *timeout* seja 0 e não exista nenhuma mensagem pendente para ser recebida, a primitiva bloqueia até receber uma mensagem.

- **Envio e recepção de eventos**: os processos presentes num grupo podem comunicar de forma assíncrona através de eventos. Os eventos são enviados por um

membro do grupo e têm como destino todos os membros do grupo, no entanto, serão somente notificados do evento, aqueles que tenham definido um *handler*;

  - `void setEventHandler(JGSEventHandler handler)`
    Permite definir um *handler* para o tratamento de eventos emitidos pelo grupo. Depois de definido, todos os eventos difundidos nesse grupo serão recebidos localmente no processo, através da chamada assíncrona do *handler*;
  - `JGSEventHandler getEventHandler()`
    Permite obter o *handler* definido para o tratamento de eventos emitidos pelo grupo;
  - `void sendEvent(java.io.Serializable obj)`
    Envia um evento com um determinado objecto (*obj*) para o grupo.

- **Espaço de tuplos partilhado**: as primitivas para interacção através do espaço partilhado de tuplos associado ao grupo, são baseadas no modelo Linda:

  - `Tuple out(Tuple t)`
    Insere um dado tuplo no espaço partilhado do grupo, podendo o tuplo não se encontrar totalmente instanciado. Após esta operação o tuplo fica disponível no espaço partilhado e acessível aos restantes membros;
  - `void in(Tuple t)`
    Remove um tuplo do espaço partilhado do grupo, esta primitiva devolve um tuplo existente no espaço compatível com o tuplo dado como argumento. Caso não exista nenhum tuplo compatível bloqueia até esse tuplo existir;
  - `Tuple inp(Tuple t)`
    Primitiva idêntica à anterior, mas não bloqueia, ou seja, retorna imediatamente no caso de não existir um tuplo compatível;
  - `Tuple rd(Tuple t)`
    Lê um tuplo do espaço partilhado do grupo mas sem o remover. Tal como nas primitivas anteriores, devolve tuplo compatível com o tuplo passado como argumento. Caso não exista nenhum tuplo compatível, bloqueia até esse tuplo existir;
  - `Tuple rdp(Tuple t)`
    Versão não bloqueante da primitiva *rd*;
  - `Vector rdpall()`
    Devolve um vector com todos os tuplos que existem no espaço partilhado do grupo.

A representação de um processo Java, num grupo de processos é realizada através de um objecto da classe JGroupSpace, ou seja por cada grupo a que o processo se pretenda filiar, deve ser instanciado um objecto desse tipo. Na figura 5.4 pode observar-se como é modelada a situação em que um processo pertence a dois grupos distintos.

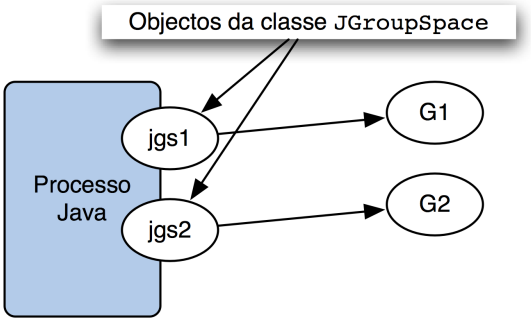

Figura 5.4: Exemplo de um processo Java que pertence a dois grupos [Cus08]

Cada grupo, no sistema JGroupSpace, é identificado por um nome (*String*) definido quando da operação de *join*. Como resultado desta operação é devolvido um objecto da classe JGSAddress, que permite identificar o processo dentro do grupo, essa classe constitui um encapsulamento de um endereço IP e de uma porta de ligação [4].

5.4 Componentes da arquitectura

As longo das secções seguintes é feita uma descrição da arquitectura de suporte ao modelo e de cada uma das suas componentes (figura 5.5). Para cada uma delas são apresentadas as suas estruturas de dados e métodos desenvolvidos, assim como os mecanismos e opções tomadas na implementação.

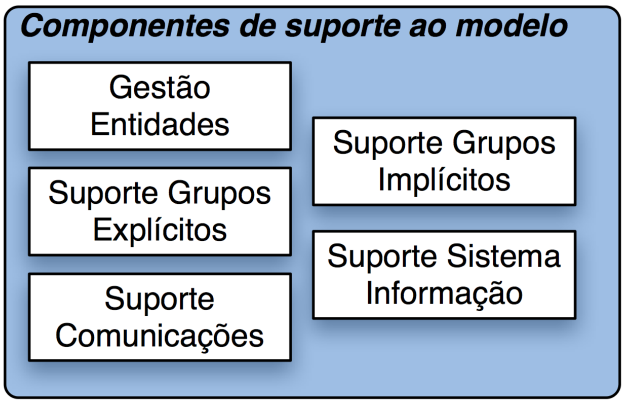

Figura 5.5: Componentes constituintes da arquitectura de suporte ao modelo

[4]Sendo este o endereço que é aqui denominado de endereço físico e que é mapeado para um endereço lógico, pelo servidor de endereços que foi definido ao nível da arquitectura MAGO.

A implementação da arquitectura aqui apresentada e que suporta o modelo MAGO, enquadrou-se no contexto de uma plataforma orientada a objectos mais particularmente em Java, pelo que algumas das opções tomadas são consequência deste facto.

Como já foi referido, esta implementação recorre à plataforma JGroupSpace e à interface de programação por esta disponibilizada, que oferece, sob a forma de objectos Java, um conjunto de mecanismos para lidar com grupos, eventos e espaço partilhado. Tendo por base estas funcionalidades foram desenhadas as várias primitivas de acordo com a semântica apresentada no modelo.

Ao longo das secções seguintes vai ser discutida a organização interna dos diversos elementos constituintes da arquitectura proposta, bem como a estrutura, a implementação e processo de execução das primitivas referidas no modelo e que são oferecidas ao programador de aplicações sob a forma de métodos de uma classe a que este tem acesso[5].

A implementação das primitivas referidas no modelo, assim como as estruturas e mecanismos mais directamente a elas associados encontra-se descrita nas secções gestão de entidades elementares e grupos e mecanismos de comunicação (secções 5.5.4 e 5.6), que constituem o núcleo central da arquitectura. Os elementos de suporte ao sistema informação e grupos implícitos, são descritos nas secções seguintes. Estes elementos, que estão mais directamente relacionados entre si, constituem uma camada de mais alto nível da arquitectura, que recorre às definições e funcionalidades disponibilizadas pelas núcleo central.

5.5 Gestão de entidades elementares e de grupos

Nesta secção vão ser apresentados os elementos base que constituem a estrutura de suporte à gestão de grupos e entidades elementares (e que se encontram representados na figura 5.6), sendo no final descrita a forma como foram implementadas as primitivas, definidas pelo modelo para a sua gestão. A gestão de grupos aqui descrita refere-se aos grupos explícitos, sendo o processo de criação dos grupos implícitos e a gestão da sua constituição analisados em detalhe na secção 5.8. As diferenças entre estes dois tipos de grupos prendem-se com a forma como são criados e como é desencadeado o processo de filiação, sendo no entanto, as restantes funcionalidades semelhantes para os dois.

Como se pode observar na figura 5.6, foi definido na arquitectura um servidor de endereços, que é o responsável pela atribuição de endereços únicos e estáticos às entidades e grupos do sistema. A tarefa deste servidor consiste em efectuar um mapeamento entre os endereços atribuídos pela plataforma JGroupSpace e endereços lógicos absolutos no sistema, o que permite isolar o problema da atribuição de endereços. Os

[5] A classe aqui referida foi denominada de MagoEntity.

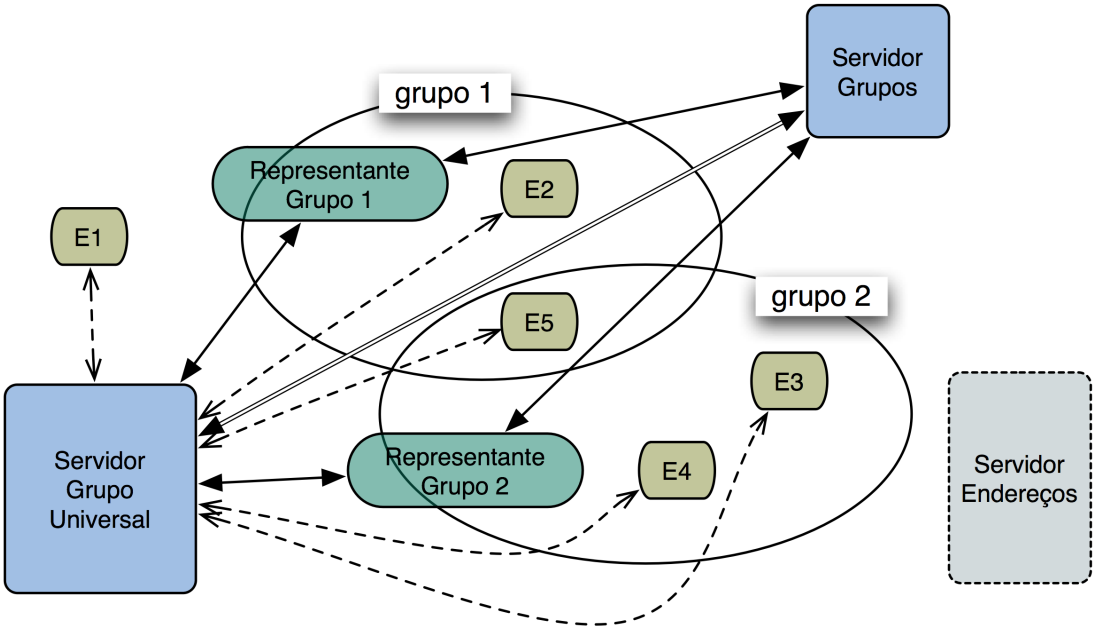

Figura 5.6: Organização dos elementos da arquitectura do sistema, responsáveis pela gestão de entidades

endereços atribuídos pelo JGroupSpace não são estáticos, ou seja, é atribuído um novo endereço sempre que o elemento entra no sistema, quer por ser um novo elemento que se regista, quer por ter reiniciado a sua execução (por exemplo, por a entidade se ter desligado temporariamente do sistema por falta de bateria do seu dispositivo computacional de suporte). É garantido que os endereços lógicos absolutos, atribuídos pelo servidor de endereços, são únicos e existe uma correspondência directa entre esses endereços e os endereços atribuídos pela plataforma JGroupSpace. Estes endereços lógicos são estáticos, ou seja, após a sua atribuição não sofrem modificações, sendo da responsabilidade do servidor garantir as funcionalidades que permitam efectuar a reconfiguração do mapeamento entre estes endereços e os da plataforma.

O servidor de endereços é responsável por:

- gerir a atribuição de endereços lógicos;
- efectuar o mapeamento entre os endereços lógicos absolutos e os endereços físicos (atribuídos pelo JGroupSpace);
- responder a pedidos de informação referentes aos endereços de uma dada entidade elementar ou grupo.

Interagem com este elemento do sistema as entidades elementares (E) e os representantes de grupo, sempre que necessitem de informação relativa aos endereços que se encontram atribuídos.

Este servidor tem disponível a função que efectua a atribuição de endereço lógico absoluto, dado um endereço da plataforma.

newMGAddress($\underline{in}$: $identificador, endereco\_JGS; \underline{out}$: $endereco\_logico$)

O campo identificador contém a identificação da entidade e/ou grupo que efectua o pedido de endereço e tem como funcionalidade facilitar o processamento de pesquisa e atribuição de endereços por parte do servidor. O *endereco_JGS* é um endereço atribuído pela plataforma JGroupSpace e possui um formato do tipo IP/porta, sendo atribuído um novo endereço sempre que é efectuada uma operação de filiação ao nível desta plataforma.

Para efectuar a actualização do mapeamento entre endereços foi desenvolvida a função *setMGAddress*().

setMGAddress($\underline{in}$: $identificador, endereco\_JGS, endereco\_logico$)

Esta função tem como parâmetros de entrada o endereço lógico e o novo endereço físico atribuído pela plataforma; como já foi referido esta situação ocorre quando dá entrada no sistema uma entidade que já se encontrava filiada num dado grupo.

O servidor de endereços disponibiliza também funcionalidades que permitem efectuar a consulta de endereços.

- **getMGAddress**($\underline{in}$: $endereco\_JGS; \underline{out}$: $endereco\_logico$), que dado um endereço da plataforma retorna o endereço lógico que lhe está associado;
- **getJGSAddress**($\underline{in}$: $endereco\_logico; \underline{out}$: $endereco\_JGS$), que dado um endereço lógico retorna o endereço da plataforma que actualmente lhe está associado.

Na informação referente aos diversos elementos que fazem parte da organização da arquitectura do sistema, ao fazer referência a endereços está-se implicitamente a referir a endereços lógicos, dado que estes são os que se mantém inalteráveis, após a sua atribuição.

5.5.1 Grupo universal

O modelo proposto assume a existência de um grupo universal ao qual todas as entidades pertencem, como já foi anteriormente referido na descrição do modelo. Ao nível da arquitectura, esse elemento é representado pelo servidor de grupo universal (servidor GU), que tem a seu cargo a gestão das acções relacionadas com as entradas e saídas de todas as entidades elementares e grupos, assim como a manutenção das estruturas de suporte da configuração do sistema. Este é um elemento central do sistema, sendo da sua responsabilidade configurar a inicialização do sistema num determinado local.

Uma configuração do sistema é constituída por:

- um grupo universal (GU);
- conjunto de entidades elementares, registadas como membros do GU;

- conjunto de grupos explícitos;
- conjunto de grupos implícitos.

O servidor GU é o suporte de qualquer aplicação, agregando todos os elementos, entidades elementares e grupos, servindo também de mediador entre estes e o sistema de informação. Devido ao facto de pertencerem ao Grupo Universal (GU) e terem um espaço de interacção comum, as entidades podem comunicar entre si através das primitivas de interacção disponibilizadas pelo modelo e a que têm acesso.

No protótipo desenvolvido, optou-se por realizar a gestão do GU através de um único servidor. Esta opção, no entanto, não compromete uma realização do serviço de gestão do GU que evite a existência deste elemento centralizado do sistema. Em particular, este serviço pode ser realizado por um grupo de servidores distribuídos, que cooperem para assegurar as tarefas de gestão do GU.

As tarefas da responsabilidade do servidor (GU) são:

- **configuração**: configurações do sistema, como por exemplo garantir que todos os elementos do sistema possuem um identificador único;
- **gestão**: criação e remoção de entidades elementares e grupos e gestão das estruturas de dados que os suportam, e ainda a gestão das interacções entre entidades;
- **suporte**: acesso aos dados locais ao servidor (através das suas estruturas internas) e o acesso ao nível do sistema de informação, sendo o responsável por manter actualizada a informação referente às entidades elementares e aos grupos.

Este servidor tem como principal função, atender os pedidos de criação (*register* e *create*) e remoção (*unregister* e *destroy*) de entidades elementares e grupos, bem como gerir e manter actualizadas as respectivas estruturas de suporte.

Ao nível das estruturas de dados que dão suporte aos elementos presentes no sistema, existem basicamente três estruturas de dados: para as entidade elementares; para os grupos explícitos; e para os grupos implícitos. A informação referente aos grupos está relacionada com parâmetros de configuração definidos quando da sua criação, sendo esta informação associada aos representantes de grupo. Estas estruturas são actualizadas de acordo com as interacções e a dinâmica de funcionamento do sistema.

De seguida é apresentada uma descrição mais detalhada da constituição das estruturas referidas.

a) Entidades Elementares O servidor GU mantém uma lista com informação de base referente a todas as entidades elementares que efectuaram o registo no sistema. Cada entidade elementar que deu entrada no sistema, passa a ter os seus dados acessíveis na "lista de entidades" mantida por este servidor, sendo os dados, referentes à entidade, removidos somente na situação em que existe uma saída explícita da entidade

do sistema[6]. A informação contida na lista, referente a cada entidade, é composta pelos seguintes elementos:

- identificador: identificador único da entidade no sistema, que funciona como um nome de *login*;
- nome simbólico: nome atribuído pelo utilizador, que pode ser utilizado para designar a entidade, de uma forma mais descritiva, ao nível da aplicação;
- validação: campo de controlo, que funciona como chave de acesso;
- endereço universal: endereço lógico atribuído à entidade, relacionado com o serviço de comunicação da plataforma intermédia de suporte;
- data de entrada: data em que foi criada a entidade;
- modo (*status*): o modo de actividade da entidade, dentro do grupo universal (ONLINE/OFFLINE);
- tipo de entidade: campo oferecido ao nível da aplicação, utilizado por exemplo, para configurar diferentes permissões de acesso a diferentes utilizadores; este campo oferece às aplicações a possibilidade de definir diferentes tipos de utilizadores. Pode ser utilizado pelas aplicações como auxiliar na sua gestão, por exemplo para configurar diferentes permissões de acesso a diferentes tipos de utilizador;
- acesso ao sistema de informação (SI): identificação da informação que representa a entidade no sistema de informação (SI), esta informação pode ser, por exemplo um endereço sendo dependente do suporte de SI escolhido.

b) Grupos Explícitos O servidor de GU mantém uma estrutura de suporte à gestão de grupos explícitos, onde cada grupo, ao ser criado, tal como no caso das entidades elementares, passa a ter uma entrada na lista de grupos existentes. A informação genérica referente a cada grupo, contém os seguintes elementos:

- identificador do grupo (id_grupo): identifica, de forma única, o grupo no sistema;
- nome simbólico: nome atribuído pelo criador do grupo, com o objectivo de melhor caracterizar o grupo, sendo definido quando da sua criação;
- identificador do criador: identificador da entidade que desencadeou o processo de criação do grupo;
- endereço universal: endereço pelo qual o grupo é visto dentro do espaço do GU, sendo este endereço atribuído pelo servidor de endereços;

[6] A saída é desencadeada por uma acção explícita por parte da entidade.

- endereço local: endereço pelo qual o grupo é visto pelos seus membros, sendo este endereço atribuído pelo servidor de endereços;

- máximo número de membros: número máximo de membros que podem filiar-se neste grupo;

- tipo: campo definido pela aplicação, por exemplo para gerir diferentes permissões e serviços;

- tipo de acesso: política de filiação seleccionada para o grupo, existindo definidas e implementadas diversas políticas que podem ser escolhidas quando da criação do grupo, sendo a execução destas da responsabilidade do elemento Representante de Grupo[7];

- data de criação: data da criação do grupo;

- acesso ao sistema de informação (SI): identificação da informação que representa o grupo no sistema de informação (SI), esta informação, tal como no caso das entidades elementares, pode ser, por exemplo, um endereço a partir do qual podemos aceder ao dados próprios do grupo.

Como se pode observar, ao nível de cada grupo são definidos dois endereços: o endereço universal que, tal como para as entidades elementares, é utilizado por quem quer comunicar directamente com o grupo; e o endereço local que é utilizado para as interacções internas do grupo, sendo este conhecido somente pelos seus membros.

c) Grupos Implícitos Os grupos implícitos distinguem-se dos grupos explícitos, pela forma como são geridos, no que se refere à detecção e identificação dos potenciais candidatos a membro, com base nos perfis/atributos próprios destes.

Do ponto de vista da arquitectura do sistema, um grupo implícito tem por base a mesma informação que um grupo explícito, diferenciando-se destes, em termos de informação das estruturas gerida pelo servidor GU, por possuir ainda informação referente ao conjunto dos atributos que estiveram na origem da sua criação. Um grupo implícito, possui por isso, para além dos elementos já referidos anteriormente para os grupos explícitos, o campo *tags*. Este campo contém a informação referente ao conjunto de instâncias de atributos próprios, definido pela entidade que desencadeou a criação do grupo, que caracteriza o grupo implícito e que todas as entidades elementares, que dele são membros, devem satisfazer.

[7] As políticas de filiação que foram definidas/implementadas encontram-se descritas mais à frente, quando da descrição do comportamento do elemento Representante de Grupo.

5.5.2 Entidades elementares

Como já foi anteriormente referido, a realização do modelo aqui apresentada, enquadrou-se no contexto de uma modelação orientada a objectos/Java, sendo uma das principais motivações o facto da plataforma de suporte JGroupSpace, possuir uma interface de objectos Java.

Neste contexto, cada entidade elementar do sistema (E) é implementada por uma instância de uma classe, que contém o conjunto dos atributos/propriedades e estruturas que a caracteriza assim como o conjunto de métodos que permitem representar o seu comportamento enquanto elemento dinâmico do sistema.

Ao nível das propriedades da entidade, esta possui um conjunto de parâmetros que a caracterizam enquanto membro do sistema (nome, validação de acesso, etc...).

No entanto, enquanto elemento da uma aplicação, podem ser definidos por esta novos campos. Estes novos parâmetros são tratados como um conjunto de dados próprios de entidade, que é armazenado no sistema de informação, que foi configurado e definido para uma aplicação específica. Em termos de modelo de base e enquanto elemento do sistema, cada entidade mantém uma descrição dos seus atributos, sendo estes:

- identificador: da entidade no sistema;
- nome simbólico: atribuído à entidade pelo utilizador da aplicação que pode, por exemplo, traduzir o seu nome próprio;
- validação: valor de validação de acesso da entidade ao sistema, funcionando como um campo de chave de acesso;
- endereço universal: endereço atribuído à entidade no grupo universal, a partir do qual a entidade pode ser directamente contactada;
- tipo de entidade: este campo coincide com o campo do mesmo nome que existe na estrutura de suporte a entidades, no servidor GU.

Para se tornarem membros de um grupo, as entidades elementares têm que proceder a uma acção de filiação que, tendo por base a plataforma de suporte, equivale a definir/criar um objecto da classe JGroupSpace, sendo a partir deste que se acede às funcionalidades e métodos que permitem agregar-se em grupos.

Após a criação deste objecto, que representa a entidade num novo grupo, deverá ser desencadeado o processo de filiação, sendo atribuído à entidade um endereço na plataforma de suporte JGroupSpace. Tendo esse endereço, é contactado o servidor de endereços de modo a obter o endereço lógico absoluto da entidade nesse novo grupo.

Cada entidade elementar é representada por um objecto da classe *JGroupSpace Object* e um endereço lógico absoluto, por cada grupo do qual seja membro.

Como já foi referido, todas as entidades são membros do grupo universal, pelo que cada entidade possui informação referente à sua filiação neste grupo assim como dos restantes grupos em que se filiou (explícitos e implícitos).

- informação referente ao GU:
  - objecto do grupo universal: ao nível da plataforma intermédia é criado um objecto do tipo JGroupSpace, que associa a entidade ao grupo universal;
  - número de grupos: número de grupos de que a entidade é membro, para além do grupo universal;
  - endereço universal.
- lista com a informação relativa à sua filiação nos seus diversos grupos, tendo cada entrada nessa lista a seguinte informação:
  - identificador do grupo (id_grupo);
  - endereço local do grupo: ou seja o endereço local do representante de grupo (descrito mais à frente);
  - objecto do grupo: ao nível da plataforma intermédia é criado um objecto do tipo JGroupSpace, por cada novo grupo a que a entidade se associa;
  - endereço local da entidade no grupo: ao filiar-se num grupo é atribuído à entidade um novo endereço lógico absoluto (recorrendo ao servidor de endereços), através do qual esta é contactada dentro do contexto do grupo;
  - modo: estado actual da entidade nesse grupo (ONLINE/OFFLINE).

Cada entidade tem também definida uma estrutura de suporte aos eventos[8] e à forma de lidar com estes, constando dessa informação:

- tipo de evento: identificador de evento que subscreveu;
- método de atendimento: nome do método que traduz o comportamento a ser desencadeado, quando a entidade recebe a notificação de ocorrência de um dado tipo de evento;
- produtor: indicação de se a entidade é produtora desse tipo de evento ou não.

Resumindo, uma entidade, na implementação da arquitectura, é representada por uma classe que agrega informação referente aos dados próprios da entidade, aos grupos a que pertence, e aos tipos de eventos que subscreveu em cada grupo.

[8]Os eventos e a forma de lidar com estes são descritos, quando da descrição dos mecanismos de comunicação.

5.5.3 Grupos

Os grupos, ao nível da arquitectura, são geridos por dois elementos: um servidor de grupos que tem a seu cargo a criação de novos grupos e o representante de grupo. Este último é criado quando da criação de um novo grupo e actua como seu "procurador", gerindo as filiações e comunicações desse mesmo grupo. O servidor de grupos actua como um gestor de representantes de grupo, gerindo o "lançamento" de representantes por cada novo grupo que é criado.

São de seguida descritos estes dois elementos, responsáveis pela gestão de grupos, assim como as políticas de filiação a que cada grupo pode obedecer.

Servidor de Grupos

Este elemento do sistema tem, como função, todo o processo de gestão da criação de novos grupos. Tal como já foi referido para o servidor de grupo universal, optou-se aqui também, por realizar a gestão de grupos através de um único servidor.

O servidor de grupos interage directamente com o servidor do grupo universal (GU) quando é efectuado um pedido de criação de grupo por parte de uma entidade elementar, sendo da sua responsabilidade a actualização dos dados referentes a cada grupo no servidor GU, assim como lidar com a destruição do grupo.

O servidor GU recebe pedidos de criação de grupo por parte de uma dada entidade, valida e processa os dados, cria uma entrada na estrutura de grupos e no sistema de informação para o novo grupo e procede à actualização dos dados. De seguida efectua um pedido de criação de grupo ao servidor de grupos, que procede à geração de um novo grupo. Neste contexto é lançado um fluxo de execução (*thread*) que corresponde a executar uma instância do programa do representante de grupo. A instância lançada é configurada de acordo com a parametrização especificada, quando da chamada da primitiva de criação do grupo.

Em termos de funcionamento geral, este elemento recebe pedidos de criação de grupos, consulta e valida os dados através de troca de informação com o servidor GU. No caso de ser válida a criação, gera o novo grupo (através da criação do seu representante) e envia a actualização de dados ao servidor GU.

Representante de Grupo

O Representante de Grupo é um processo que gere o comportamento da entidade "grupo", sendo responsável pelas interacções do grupo, ao nível dos acessos e das comunicações com este. O representante de grupo é gerado quando da criação do grupo[9], definindo o fluxo de execução a ser cumprido pelo "grupo" enquanto entidade computacional.

[9]O seu fluxo de execução é activado pelo servidor de grupo, quando da chamada da primitiva *create*.

Enquanto funções, o representante de grupo, é responsável por:

- manter uma estrutura com a lista dos membros admitidos no grupo e o estado destes (ONLINE/OFFLINE);
- gerir a execução das políticas de filiação no grupo;
- gerir o processo de destruição do grupo;
- mediar as comunicações no grupo.

Na implementação um representante de grupo é caracterizado pela sua representação no grupo universal e no seu grupo local, que é suportada, ao nível da plataforma intermédia por duas instâncias distintas da classe JGroupSpace. Paralelamente é atribuído, pelo servidor de endereços, um endereço em cada uma dessas instâncias (figura 5.7), tendo por base os endereços atribuídos pela plataforma.

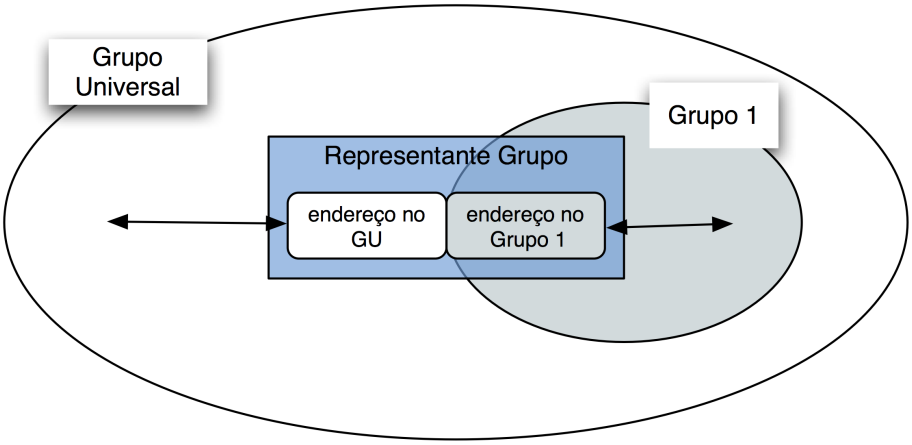

Figura 5.7: Endereços de um representante

Um destes endereços (endereço universal) permite efectuar interacções, do representante de grupo, enquanto elemento do sistema (ou seja enquanto membro do grupo universal), sendo este utilizado para a comunicação com os elementos externos ao grupo que representa. O outro, endereço local, é definido para dar suporte às comunicações internas, ou seja entre os membros do grupo, sendo este endereço conhecido e acessível somente aos membros.

O representante de grupo possui definidas estruturas que lhe possibilitam a manutenção de informação referente às suas características, atributos e estado actual. Essa informação, constitui em parte uma réplica da informação referente ao grupo, existente no servidor GU, sendo a restante referente ao estado actual do grupo no que respeita à sua constituição, eventos actualmente suportados e estado das comunicações.

- identificador do grupo (id_grupo);

- nome simbólico do grupo;
- identificador da entidade que o criou;
- endereço universal: atribuído pelo servidor de endereços quando da sua criação;
- endereço local: atribuído pelo servidor de endereços quando da sua criação;
- tipo de acesso: política de filiação a ser observada pelo grupo;
- data de criação;
- acesso ao sistema de informação (SI);
- lista de identificadores dos membros filiados e do seu estado de actividade no grupo (ONLINE/OFFLINE);
- lista de tipos de eventos definidos para o grupo com nome da entidade anunciante e método de atendimento especificado;
- filas de mensagens: estrutura auxiliar das comunicações directas com o grupo.

Os últimos três campos descrevem a informação referente ao estado actual do grupo no que se refere à sua constituição, eventos suportados e comunicações em curso.

São de seguida apresentados os passos do ciclo de vida de um representante de grupo, após ser "lançado" pelo servidor de grupos.

a) fase de inicialização do grupo e das suas estruturas de suporte:

1. criação de uma instância da classe JGroupSpace, através da qual efectua a comunicação no grupo universal (com atribuição do respectivo endereço nesse grupo);
2. obtenção de um endereço universal lógico absoluto através do servidor de endereços;
3. criação duma instância da classe JGroupSpace, através da qual efectua a comunicação no seu grupo local, ou seja para o grupo que está a ser criado;
4. obtenção de um endereço local lógico absoluto através do servidor de endereços;
5. actualização dos elementos referentes ao novo grupo, na estrutura de suporte a grupos (definida no servidor GU);
6. definição e activação da recepção de eventos definidos e suportados pelo grupo;

7. criação/actualização dos dados referentes ao grupo;
8. confirmação para a entidade que efectuou o pedido de criação do grupo, validação da sua criação ou indicação de erro.

b) ciclo de execução:

1. aguarda chegada de mensagem do exterior;
2. processa mensagem.

Como já foi referido, um representante de grupo tem definidos dois objectos distintos para interacção (na plataforma JGroupSpace) e consequentemente dois endereços (passos 1 a 4 do processo de inicialização), sendo o primeiro (endereço universal) definido para a interacção com o exterior e o segundo (endereço local) necessário para as interacções internas do grupo (figura 5.7).

As mensagens com origem no interior do grupo são processadas através dos mecanismos de eventos do grupo. Embora exista, disponibilizada pela plataforma de suporte, a possibilidade de comunicação directa entre os membros do grupo e o seu representante através do seu endereço local, este mecanismo não é utilizado ao nível do modelo, por opção na concepção da arquitectura MAGO. Se pretenderem comunicar directamente com o representante do grupo, os membros fazem-no através do espaço de comunicação do grupo universal, utilizando o endereço atribuído ao representante no grupo universal.

Ao enviar mensagens para o grupo, estas são colocadas numa estrutura interna[10] gerida pelo representante do grupo. É da responsabilidade dos representantes efectuarem a leitura da mensagem para posteriormente efectuarem o seu processamento.

Após a sua criação, o representante de grupo lida com as seguintes mensagens do exterior e que são dirigidas ao grupo:

1. pedido de filiação no grupo *join*();
2. pedido de saída do grupo *leave*();
3. pedido de informação sobre os membros filiados no grupo e o seu respectivo estado de actividade *membership*();
4. envio de mensagem para o grupo através de comunicação directa *send*();
5. pedido de informação referente a eventos suportados no grupo, desencadeada pela primitiva *chk_advertise*();
6. pedido de destruição do grupo, que pode obedecer a diversas políticas de notificação dos membros, *destroy*().

[10] Esta estrutura será explicada mais adiante quando da descrição dos mecanismos de comunicação.

Nos tipos 3 e 5 de mensagens referidos, o representante efectua um processo simples de envio directo para a entidade que desencadeou o pedido, de uma mensagem com a lista com os valores pedidos.

No caso do tratamento de uma comunicação directa (*send*()) cujo destinatário é um grupo, podem ser tomadas diferentes políticas de divulgação da mensagem para o grupo, devendo ser lançado um evento de chegada de mensagem ao grupo, no caso de política de divulgação escolhida ser do tipo NOTIFY ou SPREAD.

Na situação de chegada de mensagem de destruição de grupo, o representante efectua o processamento necessário de modo desencadear a política indicada na invocação da primitiva *destroy*(). Existem três possibilidades de destruição de grupo: NO_NOTIFICATION, NOTIFICATION e NOTIFICATION_ACK, adiante descritas quando da análise da primitiva que desencadeia este processo (*destroy*()).

No caso da chegada de um pedido de filiação (*join*()) ao grupo, é desencadeado o lançamento de um fluxo de execução (*thread*), responsável por gerir o pedido de admissão de acordo com a política de filiação escolhida para o grupo. Somente após o envio directo de uma mensagem de confirmação de entrada, por parte do representante do grupo, para a entidade candidata, esta pode efectivar a filiação no grupo, através da chamada da respectiva primitiva de filiação da interface disponibilizada pelo JGroupSpace. Se a filiação foi efectuada com sucesso, o representante retorna o endereço que a entidade passa a ter no espaço do novo grupo e actualiza a sua estrutura de suporte de grupo.

Políticas de filiação

Em termos de políticas de admissão no grupo, foram definidas diferentes formas de filiação que podem ser escolhidas pela aplicação, de acordo com as características que esta pretenda para o grupo. A política de admissão que vai ser seguida pelo grupo, é definida (escolhida) quando da criação do grupo, podendo nessa altura optar-se por: (1) acesso aberto; (2) acesso com lista participantes; (3) votação por maioria; (4) votação por um membro; (5) votação pelo criador do grupo. A forma de realização destas diferentes políticas é apresentada de seguida.

- ***(1) Acesso aberto***
  Nesta situação todos os candidatos a membros são aceites[11], não passando por nenhum processo de selecção. O representante de grupo deve proceder à actualização da lista de membros, criando uma nova entrada e enviando uma mensagem de aceitação à entidade, que pode então efectivar a sua filiação no grupo e actualizar os seus dados nas respectivas estruturas.

[11] No caso do grupo ter definido um número máximo de membros, os candidatos são aceites enquanto este valor não for excedido.

- ***(2) Lista de participantes***
  Neste caso, ao criar o grupo, é passada uma lista com as permissões de acesso ("senhas de acesso") ao grupo, sendo esta lista gerida pelo representante do grupo. Ao efectuar o pedido de admissão, cada entidade deve fornecer a sua senha (uma por entidade e sem repetições), para que possa ser validada a sua candidatura a membro.

  Do lado do representante, este recebe o pedido por parte da entidade, com a informação de acesso, verificando depois se esta senha é válida, ou seja, se pertence à lista e não tiver sido ainda utilizada:

  – se for válida: marca a senha como utilizada e procede à actualização da lista de membros;

  – se não for válida: recusa a filiação do candidato.

  No caso de já ter sido utilizada, é assumido que o candidato já procedeu à sua filiação.

- ***(3) Votação por Maioria***
  Na implementação das políticas de filiação por votação, é utilizado ao nível do sistema, o espaço de tuplos como mecanismo de "recolha" de votos. Nesta situação, o representante, após a recepção do pedido de filiação, desencadeia o processo de votação recorrendo às funcionalidades de manipulação do espaço de tuplos, que são oferecidas pela camada JGroupSpace[12].

  A sequência de acções executadas, por parte do representante, para a votação, é a seguinte (ver figura 5.8):

  – geração do tuplo de voto (urna de votos), por parte do representante;

  – dá início ao processo de votação:

    (a) escrita do tuplo no espaço
    (formato do tuplo: $< vote\_tuple, id\_candidate, (numMembers/2) + 1 >$);

    (b) publicação do evento de votação, ou seja difusão pelos membros do grupo, de uma mensagem com informação referente à votação em curso;

  – leitura com remoção do tuplo do espaço ($< vote\_tuple, id\_candidate, 0 >$), ou seja, aguardar a existência de um tuplo no espaço que verifique estas condições;

  – após proceder à leitura (quer dizer que o campo de votos chegou a 0), actualiza a estrutura do grupo e envia uma mensagem de aceitação ao candidato.

[12]Note-se que a este nível, o espaço de tuplos utilizado não é directamente visível ao nível das primitivas do modelo MAGO, sendo aqui utilizado para suportar a implementação desta funcionalidade internamente à arquitectura.

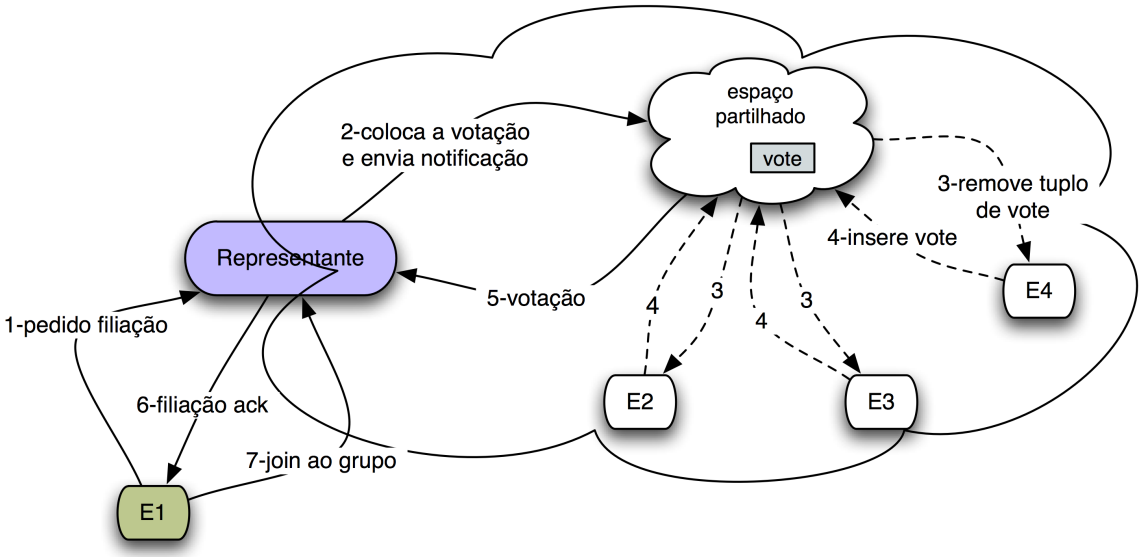

Figura 5.8: Sequência de um processo de votação

Por parte dos membros filiados no grupo, ao receberem a notificação de votação em curso e no caso de pretenderem votar favoravelmente a entrada do novo membro, executam a seguinte sequência:

- recebem notificação de votação (evento JOIN_EV) com os dados referentes a esta;
- procedem à leitura do tuplo com o formato: $< vote\_tuple, id\_candidate, \_ >$, ou seja, o tuplo $< vote\_tuple, id\_candidate, value >$ é retirado do espaço;
- actualizam o valor da votação com:
  - $value = value - 1$, no caso de votação favorável;
  - $value = value$, no caso de votação desfavorável;
- escrevem, no espaço, o tuplo actualizado: $< vote\_tuple, id\_candidate, value >$, ficando a votação (tuplo) disponível para outro membro.

Se um membro, ao desencadear o seu processo de votação, remover do espaço um tuplo com $valor <= 0$, tal significa que a votação já foi concluída, pelo que deve limitar-se a repôr o tuplo no espaço.

O processo de admissão por votação tem um limite de tempo definido, ou seja um tempo durante o qual o representante fica a aguardar pela conclusão da votação; se este tempo for excedido[13], a votação é cancelada, sendo enviado para o candidato o aviso de recusa de entrada. O tuplo referente à votação é então removido do espaço pelo representante e colocado um novo tuplo com o $valor = -1$.

[13]O tempo pode ser excedido por ocorrência de falhas ou por membros se encontrarem *offline*.

Desta forma, é possível também verificar se um dado candidato já foi sujeito a votação ou não, ou seja, antes de colocar um candidato a votação, o representante efectua uma leitura prévia do tuplo: $< vote\_tuple, id\_candidate, \_ >$. Se o tuplo existia no espaço, tal significa que essa entidade já foi sujeita anteriormente a um processo de votação no grupo, encontrando-se o processo de admissão concluído.

De salientar que, no processo de votação por maioria, o mecanismo de votação só é desencadeado após a existência de um membro no grupo, sendo atribuído a esse primeiro membro um acesso directo, ou seja sem passar por um processo de votação. Note-se que, o representante de grupo não é contabilizado como membro de grupo nestas contagens.

- ***(4) Votação por 1***
  Este mecanismo de admissão de membros é em tudo idêntico ao anteriormente descrito, tendo, neste caso, o campo valor do tuplo de votação, um valor inicial de 1.

  Formato do tuplo: $< vote\_tuple, id\_candidate, 1 >$.

  Neste caso, a admissão do primeiro membro é também directa, já que não existem ainda membros que possam validar a sua entrada.

- ***(5) Admissão autorizada pelo criador***
  Neste caso, a permissão de entrada de novos membros é concedida por uma entidade especial, a criadora do grupo. O processo de admissão segue a seguinte sequência de acções:

  – representante envia, à entidade criadora do grupo, uma mensagem com o pedido de admissão e com os dados do candidato;
  – fica a aguardar a resposta do criador;
  – no caso de uma resposta positiva, procede à actualização da lista de membros do grupo e comunica ao candidato que pode efectivar a sua filiação.

  Do lado da entidade criadora, existe um método de recepção de pedidos de admissão, que é invocado directamente pelo representante, quando efectua o pedido. Se a entidade criadora tiver saído do sistema ou se encontrar OFFLINE, existe definido um tempo limite para o processo de admissão, ou seja, se o tempo for excedido a autorização é negada, sendo enviado para o candidato o aviso de recusa de entrada.

5.5.4 Primitivas de gestão de entidades elementares e de grupos

São de seguida descritas com maior detalhe a semântica operacional e a realização das primitivas oferecidas pelo modelo e que são definidas como métodos que permitem

representar o comportamento das entidades, no que se refere à sua gestão própria e enquanto elementos dinâmicos do sistema.

São inicialmente apresentadas as primitivas que efectuam a manipulação de entidades elementares enquanto membros do grupo universal, efectuando a sua gestão e a sua identificação única, como membros do sistema. Finalmente são apresentadas as primitivas que permitem efectuar a gestão de entidades dentro da estrutura de grupos.

- **Registo de entidades no sistema -** *register*()
  Esta primitiva regista a entrada de entidades no sistema. Podem verificar-se duas situações distintas:

  – tratar-se de uma entidade "nova", tem que ser garantida a atribuição de um identificador único e estabelecido o seu "espaço" de comunicação no grupo universal;

  – tratar-se de uma entidade já pertencer ao sistema, tem que se proceder a uma reatribuição de endereços e efectuar a respectiva actualização nas estruturas de suporte do sistema.

  No primeiro caso a entidade não possui ainda um registo, ou seja, é a sua primeira interacção com o sistema (ver figura 5.9):

  – desencadeia o pedido de acesso, comunica com o servidor GU, através de RMI (*Remote Method Invocation*), chamando métodos do servidor para criar uma entrada, nas estruturas e no SI, para a nova entidade, sendo-lhe retornado o identificador atribuído;

  – cria objecto (JGroupSpace) e filia-se no grupo universal ("GroupU" é o identificador atribuído ao grupo universal) através da primitiva disponível na plataforma de suporte (JGroupSpace); desta forma, passa a ter acesso aos mecanismos de interacção definidos para este tipo de objecto;

  – comunica, ao servidor GU, os dados referentes à sua filiação (endereço universal), através do mecanismo de comunicação directa.

  No segundo caso, a entidade já pertence ao sistema (ou seja, já tinha efectuado uma acção prévia de registo), trata-se de efectuar uma operação de "login", isto é iniciar uma nova "sessão". Nesta situação tem que se proceder a uma actualização do mapeamento de endereços, ao nível do servidor de endereços, para cada um dos grupos em que se encontra filiada.

  – cria objecto (JGroupSpace) e obtém novo endereço na plataforma[14];

[14]Os endereços atribuídos na plataforma JGroupSpace não são fixos, sendo atribuído um novo endereço de cada vez que é efectuada a acção de criação do objecto que representa a entidade no grupo.

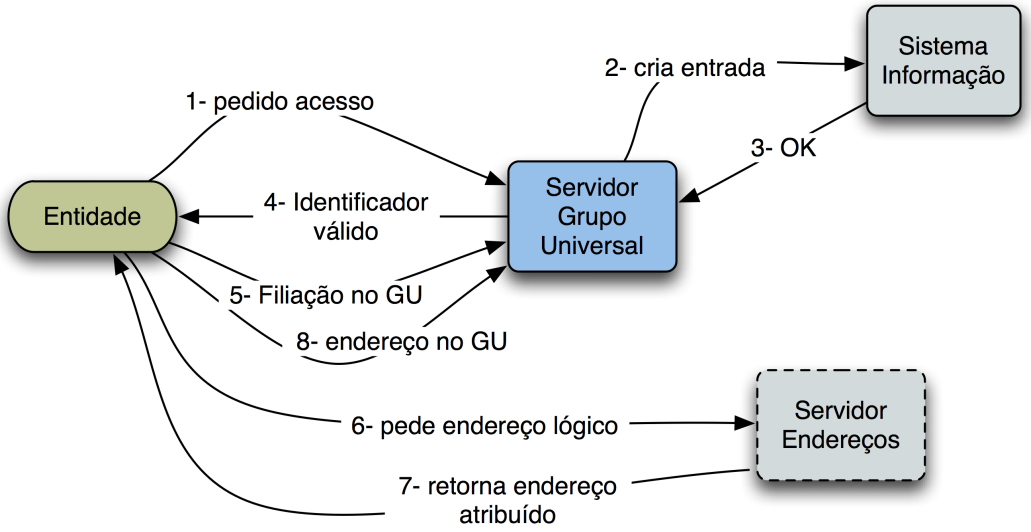

Figura 5.9: Sequência de acções desencadeada pela invocação da primitiva *register*()

- actualiza o mapeamento deste endereço no servidor de endereços (*setMGAddress*());
- percorre a sua estrutura de grupos e desencadeia o processo de criação e junção nos seus grupos, e respectiva actualização de endereços no servidor de endereços. Este processo é distinto do de filiação, pois a entidade não é novamente sujeita a um processo de admissão, procedendo somente a uma actualização do mapeamento dos endereços lógicos no servidor de endereços.

Na fase de registo da entidade foi também utilizado o campo *validação*, por questões operacionais, como forma de efectuar uma validação de acessos autorizados. A existência deste campo foi motivada pela necessidade de diferenciar as situações de entrada no sistema e a de reentrada (tipo *login*). Ao entrar no sistema foi definido um identificador único para a entidade e um endereço através do qual esta pode ser contactada no grupo universal (*login*) e foi definido também um campo de validação que actua como chave de acesso do sistema.

A sequência de operações que é desencadeada (utilizando o campo *validação*), no caso de entrada de elemento no sistema e na situação em que o *identificador* da entidade já existe, ou seja, existe um elemento na lista de entidades registadas, com o *identificador* indicado, quando do *login* da entidade:

- verifica o valor do campo "validação" (que funciona como um chave de acesso):
  - se o valor não foi definido (valor "vazio"), significa que se trata de uma nova entidade e que esta tem que se registar no sistema, procedendo ao registo normal no sistema;

  - se o valor se encontra já definido (ou seja, não "vazio"), significa que se trata de uma entidade já registada no sistema; nessa situação tem-se os seguintes casos:
    - o valor do campo de validação não coincide com o valor introduzido quando da nova entrada ("login" no sistema), nesse caso o acesso da entidade ao sistema é recusado;
    - o valor do campo de validação coincide com o valor introduzido quando da nova entrada, nesse caso o acesso da entidade ao sistema é autorizado, sendo efectuada a actualização do mapeamento de endereços ao nível do servidor de endereços

- **Remoção de entidades do sistema -** *unregister*()
  Ao invocar esta primitiva, a entidade efectua uma comunicação directa com o servidor de GU, sinalizando a sua intenção de saída do sistema. Ao receber este pedido de saída, o servidor remove a entrada da entidade da sua estrutura de suporte de entidades, remove do sistema de informação, os dados associados à entidade[15] e seguidamente, desencadeia o processo de saída da entidade dos vários grupos a que pertence, percorrendo a sua lista de grupo e desencadeando a acção de *leave* para cada um deles, ou seja, efectua uma saída explícita de grupo.

- **Criação de grupos -** *create*()
  Esta é a primitiva de criação de grupo, cuja operação é desencadeada por uma entidade registada no sistema. O conjunto de acções desencadeadas por esta primitiva inicia-se, tal como no processo de registo de uma nova entidade, com a comunicação com o servidor de grupo universal (GU), para a obtenção de um identificador único para o novo grupo. Do lado do servidor GU, ao receber o pedido de criação de grupo, informa o servidor de grupos de que deve proceder ao "lançamento" do representante de grupo.

  O conjunto de acções desencadeadas pelo representante, já anteriormente descritas, consiste da criação do objecto de comunicação do grupo universal e do novo grupo e sua associação a ambos. Procede de seguida à configuração do grupo, de acordo com os dados que foram passados como argumentos na invocação da primitiva tais como: o tipo de acesso e o número máximo de elementos admitidos.

  A sequência de interacções e os elementos envolvidos na criação de um grupo podem ser observados na figura 5.10.

- **Destruição de grupo -** *destroy*()
  O processo de destruição de um grupo é desencadeado por uma entidade pertencente a este, que comunica directamente com o representante do grupo. O re-

[15] Esta remoção pode ser somente uma sinalização, ao nível do sistema de informação, de que a entidade deixou de pertencer ao sistema.

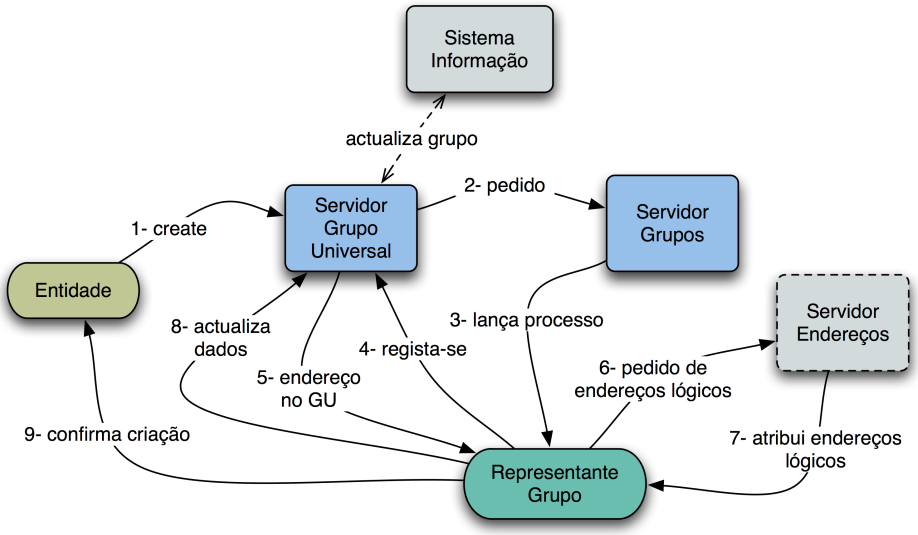

Figura 5.10: Sequência de acções desencadeadas pela invocação da primitiva *create*()

presentante, ao receber o pedido da entidade, executa o processo de destruição, de acordo com a política indicada. Foram definidos três tipos distintos, que estão relacionados com a forma como o representante do grupo procede à notificação de destruição aos membros do grupo:

- NO_NOTIFICATION:
  - inibe a difusão de eventos associados ao grupo;
  - comunica directamente com o servidor GU, para que este proceda à remoção do grupo das estruturas de suporte de grupo;
  - destrói o processo do representante do grupo.
- NOTIFICATION:
  - publica o evento de destruição do grupo, não sendo exigido nenhum comportamento específico por parte dos membros (por exemplo, invocar primitiva *leave*());
  - inibe a difusão de eventos associados ao grupo;
  - comunica directamente com o servidor GU, para que este proceda à remoção do grupo, nas estruturas de suporte de grupo;
  - destrói o processo do representante do grupo;
  - termina a execução.
- NOTIFICATION_ACK:
  - publica o evento de destruição do grupo com informação referente a este facto. Os membros do grupo, ao receberem esta notificação, devem proceder à saída explícita do grupo, através da invocação da primitiva *leave*();

    - inibe a notificação de novos eventos associados ao grupo;
    - aguarda, até que a primitiva de *membership*() retorne uma lista vazia (ou o tempo de espera seja excedido);
    - comunica directamente com o servidor GU, para que este proceda à remoção do grupo, nas estruturas internas de suporte de grupo;
    - destrói o processo do representante do grupo.

  As entidades possuem métodos de atendimento para este tipo de evento, que lhes permite executar o comportamento pretendido, pela aplicação.

- **Filiação num grupo -** *join*()
  A filiação de uma entidade num grupo é executada pela primitiva *join*(), invocada pela entidade, que desencadeia a seguinte sequência de acções (figura 5.11):

  - com base no identificador do grupo, consulta o servidor GU, para obtenção do endereço universal do grupo, ou seja, do endereço do seu representante no GU;
  - comunica directamente com o representante de grupo, através do endereço universal, enviando uma mensagem de pedido de filiação;
  - aguarda resposta por parte do representante, sendo da responsabilidade deste desencadear o processo de admissão do candidato, de acordo com a política de filiação configurada para o grupo (tipo de acesso). As políticas de filiação permitidas foram já anteriormente descritas na subsecção 5.5.3;
  - se a resposta é negativa:
    - a filiação falha;
  - se a resposta é positiva:
    - cria objecto de comunicação para o novo grupo;
    - torna efectiva a junção ao grupo, por chamada do correspondente método da plataforma intermédia, JGroupSpace;
    - obtém endereço lógico através do servidor de endereços;
    - activa a recepção de eventos do grupo ("handler");
    - comunica o seu endereço no novo grupo ao representante, para que este proceda à actualização da sua constituição;
    - preenche estruturas de dados de suporte ao nível da entidade.

- **Saída de grupo -** *leave*()
  Esta primitiva trata da remoção de uma entidade, de um grupo do qual é membro, através da seguinte sequência de acções (figura 5.12):

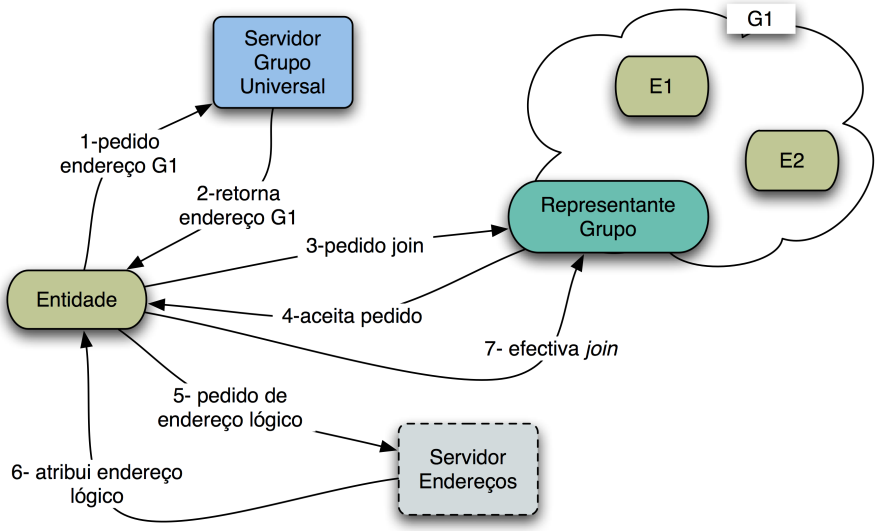

Figura 5.11: Sequência de acções desencadeada pela invocação da primitiva *join*()

- com base no identificador do grupo, consulta o servidor GU, para obtenção do endereço universal do grupo;
- comunica directamente com o representante de grupo através do endereço universal, enviando uma mensagem de pedido de remoção;
- aguarda resposta do representante;
- se a resposta é negativa:
  - a remoção falha, por exemplo, por o membro não pertencer ao grupo;
- se a resposta é positiva, o representante remove o membro da sua estrutura de membros:
  - efectua a chamada da primitiva (*leave*()) disponibilizada pelo JGroupSpace, que efectiva a remoção do elemento do espaço do grupo;
  - actualiza as estruturas de suporte da entidade, removendo a informação referente ao grupo.

- **Obtenção dos membros de um grupo -** *membership*()
  A chamada desta primitiva retorna a lista de participantes do grupo e o seu estado (modo), no momento da chamada:
  - a entidade comunica directamente com o representante de grupo, através do seu endereço absoluto, enviando uma mensagem de consulta de membros;
  - aguarda uma mensagem, no espaço do grupo universal, por parte do representante;
  - a lista enviada pelo representante retorna uma lista de elementos do tipo: $(identificador\_da\_entidade, estado\_da\_entidade\_no\_grupo)$.

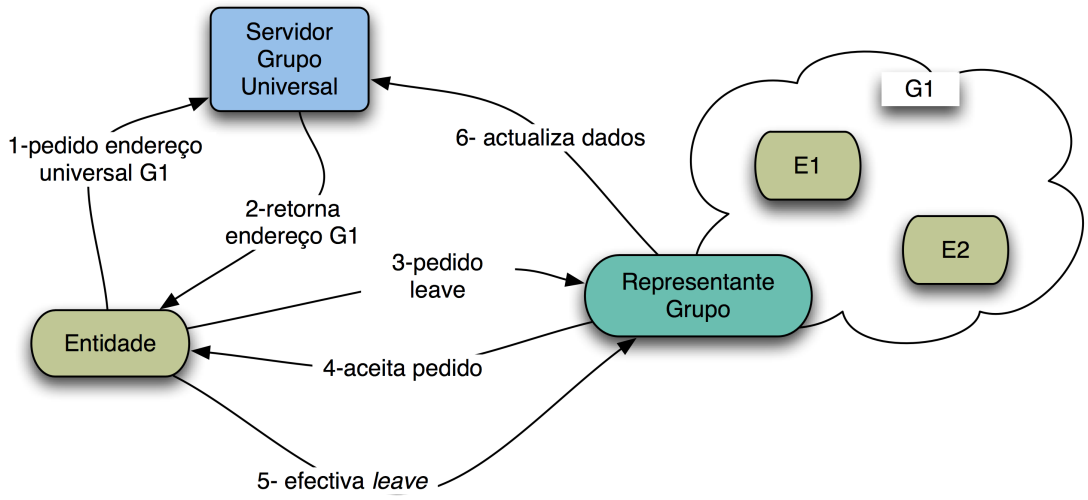

Figura 5.12: Sequência de acções desencadeada pela invocação da primitiva *leave*()

Do lado do representante de grupo, esta acção desencadeia o processo de consulta e comunicação directa com a entidade, através do seu endereço universal.

- **Alteração de estado de entidade num grupo -** *set_mode*()
  Esta função permite alterar o estado de actividade da uma entidade no grupo. Foram definidos dois estados possíveis para as entidades nos grupos a que pertencem e que representam a sua acessibilidade no interior do grupo: ligado (ONLINE); e desligado (OFFLINE).

  Ao filiar-se num novo grupo, a entidade é colocada no modo ONLINE, podendo posteriormente este estado ser alterado.

  A primitiva desencadeia a publicação de um evento de modificação de estado sobre o espaço do grupo, sendo este evento tratado pelo representante do grupo, o que tem como única função modificar o estado associado à entidade na estrutura de membros do grupo. Na mensagem do evento, é enviado o identificador da entidade e o seu estado (modo).

5.6 Mecanismos de comunicação

Ao longo desta secção é descrita a forma como foram implementados os diversos mecanismo de comunicação definidos no modelo e os métodos que os suportam.

5.6.1 Comunicação directa

Esta é a classe de primitivas para a comunicação directa entre entidades, onde é necessário especificar explicitamente o destinatário. Este tipo de comunicação é utilizado

para a comunicação directa entre entidades elementares, sendo também utilizado sempre que se pretende comunicar com um grupo enquanto unidade. Este tipo de interacção é implementado com base em mecanismo de comunicação por mensagens suportado pela plataforma JGroupSpace, onde o processo de leitura de mensagens deve ser desencadeado directamente pelas entidades destinatárias.

- **A transmissão -** *send*()
  O modelo apresenta as duas variantes que a primitiva *send*() pode tomar. Em ambos os casos, é enviada uma mensagem, destinada directamente à entidade cujo identificador é passado como argumento. Também em ambos os casos, é chamado o método do receptor, que processa a mensagem enviada.

  - **Utilização de um método genérico:**
    Neste caso, a sequência de acções desencadeada pela primitiva é a seguinte:

    * preparação da mensagem e atribuição de um identificador à mensagem (*id_msg*);
    * envio da mensagem para o endereço universal do destinatário, com utilização da primitiva disponibilizada pela interface do JGroupSpace, *send*().

    No caso do destinatário ser um grupo, a mensagem deve ser processada pelo seu representante, sendo desencadeado por este o processamento definido para o tipo de mensagem recebida.

    Se a mensagem for do tipo "mensagem de dados" e se a opção for a de divulgação desta por todos os membros (*send*() do tipo SPREAD), o representante de grupo deve publicar um evento de MESSAGE_EV, reenviando a mensagem recebida pelo representante pelos membros do grupo sob a forma de um evento. Se a mensagem de dados for divulgada através de um processo de NOTIFY, o representante de grupo deve criar um tuplo do tipo **message** com a mensagem que recebeu, colocar o tuplo na fila global do grupo (com informação do id_msg) e proceder à emissão de evento do tipo MESSAGE_RCV. Este tipo de evento é recebido por todos os membros activos que podem, com base na informação recebida, proceder à leitura da mensagem da fila global do grupo. A implementação da fila global do grupo foi implementada sobre o espaço de tuplos gerido internamente pela arquitectura do MAGO.

  - **Utilização de um método específico da interface:**
    Nesta situação, é chamado um método específico definido na interface do destinatário, sendo a designação do método passada como argumento da primitiva *send*().

- **A recepção -** *receive*()
  A invocação desta primitiva efectua uma leitura (com remoção) de uma mensagem da fila de mensagens associada à entidade.

  Como foi referido, no modelo, a primitiva *receive*() efectua a leitura explícita da fila de mensagens da entidade ou, no caso de ser indicado um grupo, efectua a leitura da fila global do grupo. No primeiro caso, a leitura é desencadeada pela entidade elementar e sobre a sua fila de mensagens.

  No caso da entidade explicitar o grupo, a leitura é efectuada sobre a fila global do grupo, onde a mensagem foi previamente processada pelo representante do grupo. Os membros do grupo são notificados pelo representante, quando da chegada da mensagem, isto se for especificada uma opção de notificação (NOTIFY/SPREAD).

5.6.2 Eventos

A implementação das primitivas de comunicação por eventos é suportada por mecanismos de eventos que são oferecidos pela plataforma JGroupSpaces.

Ao nível da realização do modelo, foi definida uma estrutura de suporte a diferentes tipos de eventos, que reúne a informação referente aos eventos que são propagados em cada grupo.

A estrutura de suporte de eventos, ao nível do representante de grupos, é constituída pelos seguintes elementos:

- identificador único do tipo de evento;
- identificador do produtor de evento;
- nome do método de atendimento que deve ser executado pelo destinatário quando este é notificado do evento;

Ao nível das entidades, existe definida uma estrutura com a informação de suporte de eventos referente a cada grupo a que pertencem. Nesta estrutura encontram-se registados todos os tipos de eventos definidos e programados pela aplicação.

A ocorrência de um evento é propagada, ao nível da camada inferior de sistema, por todos os membros do grupo onde esse evento foi desencadeado. O tratamento dos eventos, ou seja, o comportamento a ser desencadeado nos receptores quando da recepção de um dado evento, é definido no método de atendimento especificado quando da criação do tipo de evento. No caso de eventos do sistema[16], esse tratamento é assegurado pela implementação aqui apresentada. No caso de eventos definidos pela aplicação, é da responsabilidade desta a definição do método de atendimento e o

[16]Estes eventos encontram-se descritos mais à frente na secção 5.9.

seu conhecimento por parte das entidades que subscreveram esse tipo de evento.

Como já foi referido, na descrição da operação *join*, uma entidade, ao filiar-se num dado grupo, activa o "handler" de eventos do grupo, tendo esta acção também implícita a inicialização da estrutura de tipos de eventos definidos no grupo.

O conjunto de primitivas, através do qual é efectuado o tratamento de eventos que constitui um dos mecanismos preferenciais para comunicação dentro do grupo, é descrito de seguida:

- **Anúncio de novo tipo de evento -** *advertise*()
  Esta primitiva permite efectuar o anúncio de um novo tipo de evento, por parte de uma entidade num dado grupo de que é membro. Esta primitiva desencadeia a seguinte sequência de acções:
  - valida a entidade como membro do grupo;
  - actualiza dados na sua estrutura de suporte de eventos do grupo;
  - se o tipo de anúncio for:
    * activo (ACTIVE): emite (publica) um evento de aviso, de um novo tipo de evento para todo o grupo (ADVERTISE_ACT_EV);
    * passivo (PASSIVE): não desencadeia nenhum processo de notificação para o grupo, mas emite uma notificação de evento de actualização da tabela de eventos; este evento é tratado pelo representante do grupo, que procede à sua inserção na tabela de eventos definidos para o grupo (ADVERTISE_EV);

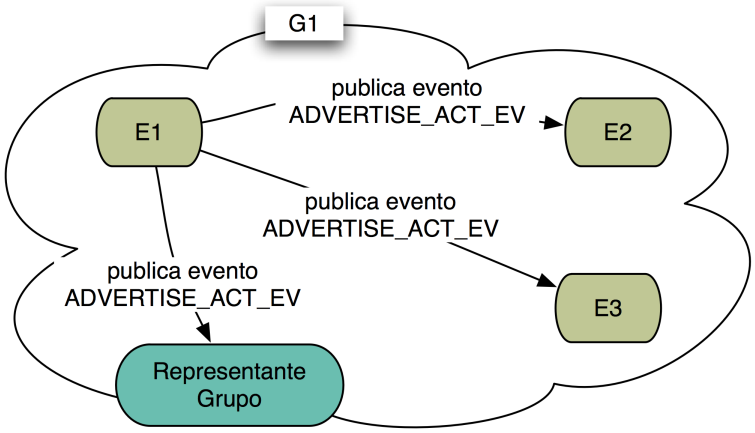

Figura 5.13: Sequência de acções desencadeada pela primitiva *advertise*() ACTIVE

Na situação de *advertise* do tipo ACTIVE, as entidades do grupo, que se encontrem activas, recebem a notificação de um novo tipo de evento e podem colocar a informação referente este, na sua estrutura de suporte de evento, através da sua

subscrição, como se pode observar na figura 5.13 onde está descrita a sequência de acções na situação de um anúncio do tipo ACTIVE. Se as entidades não estiverem activas podem, mais tarde, proceder à consulta dos tipos de eventos definidos para o grupo (*chk_advertise*) e que são geridos pelo representante de grupo.

No caso do representante do grupo, este ao receber a notificação desencadeia o processo de actualização da sua estrutura de eventos.

- **Desactivação de um tipo de evento -** *unadvertise*()
  Esta primitiva permite desactivar um dado tipo de evento, procedendo à remoção da sua definição das estruturas de suporte, da entidade produtora e do representante do grupo onde este tipo de evento foi definido.

  A entidade produtora do tipo de evento é a única que pode proceder à sua remoção, procedendo à notificação de desactivação de evento (UNADVERTISE_EV), informando deste facto o grupo, através do seu representante.

  Ao receber esta notificação, o representante de grupo procede à remoção deste tipo de evento, ou seja, procede à actualização da informação na sua estrutura de suporte de eventos. Note-se que as entidades não são notificadas da desactivação de tipos de eventos. No entanto, estas podem sempre proceder à actualização dos tipos de eventos activos através da chamada da primitiva de consulta *chk_advertise*().

- **Consulta de tipos de eventos -** *chk_advertise*()
  Esta primitiva permite efectuar uma consulta dos tipos de eventos que se encontram disponíveis, isto é, anunciados no grupo. A invocação desta primitiva é desencadeada por uma entidade membro do grupo, que procede ao envio de uma mensagem de consulta de eventos para o representante de grupo (utilizando o seu endereço universal) e fica em espera a aguardar a resposta deste com a informação pedida. O representante por sua vez, ao receber uma mensagem com este pedido, emite uma mensagem directa à entidade que efectuou o pedido, contendo a lista de todos os tipos de eventos anunciados no grupo.

- **Subscrição de um dado tipo de evento -** *subscribe*()
  A primitiva de subscrição é desencadeada por uma entidade que pretende subscrever ou proceder a uma configuração de um tipo de evento que exista definido no grupo, ou seja que foi previamente anunciado (*advertise*()) no grupo. Esta primitiva efectua a actualização da estrutura de suporte de eventos da própria entidade, de acordo com os argumentos que lhe são passados, podendo:

  – criar uma nova entrada na sua estrutura de suporte de eventos, onde é especificado qual o método de atendimento que deve ser executado, quando

da recepção para o tratamento do tipo de evento;

- **Desactivação da subscrição de uma dado tipo de evento -** *unsubscribe*()
  Esta primitiva desencadeia um processo de remoção do registo de um dado tipo de evento da estrutura de suporte de tipos de eventos da entidade receptora. Note-se que os eventos de sistema e que são automaticamente subscritos quando uma entidade se filia num grupo, não podem ser removidos.

- **Publicação de um evento -** *publish*()
  Esta primitiva é desencadeada por um produtor de evento, sendo esta a primitiva que difunde o evento para o grupo, ou seja, procede à notificação da ocorrência do tipo de evento aos membros do grupo. A difusão é efectuada através do mecanismo de eventos da plataforma de suporte JGroupSpace.

5.6.3 Espaço partilhado

Ao filiar-se num grupo as entidades passam a ter acesso a um espaço partilhado do grupo, ao qual podem aceder através do conjunto de primitivas definidas no modelo. Note-se que um mecanismo baseado em espaço partilhado de grupo é também utilizado internamente na implementação de algumas das primitivas já anteriormente descritas:

- para a implementação das políticas de filiação (*join*) que recorrem a processos de votação por parte de membros do grupo;

- para a gestão do processo de destruição/remoção de um grupo;

- para a implementação da fila global do grupo, na qual são colocadas as mensagens dirigidas ao grupo e que podem ser lidas pelos seus membros.

No entanto, este espaço interno é apenas acessível à arquitectura, não sendo visível nem acessível ao nível das primitivas do modelo.

Ao nível do modelo, as entidades elementares podem efectuar operações sobre o espaço partilhado do grupo através das primitivas descritas no modelo e que manipulam dados que são convertidos no formato de tuplos de dados.

Este conjunto de primitivas, utilizado quando se partilham dados dentro de um contexto de grupo é, de um modo geral, um processo desencadeado em duas fases:

- disponibilização dos dados no sistema de informação associados ao grupo;

- acesso ao espaço partilhado do grupo.

As primitivas agora descritas, permitem efectuar a manipulação de dados gerados pela aplicação, ao nível do espaço partilhado do grupo, sendo a gestão da informação no SI efectuada pelo módulo de acesso ao SI.

Com este conjunto de primitivas, as entidades que sejam membros de um grupo podem colocar, consultar e remover informação, sob a forma de tuplos de dados, no espaço partilhado do grupo.

Os tuplos do espaço partilhado possuem genericamente o seguinte formato:

$< nome\_tuplo, id\_produtor, tipo\_acesso, dados >$

Para a gestão do espaço partilhado interno à arquitectura, existem dois tipos de tuplos, que permitem efectuar a gestão das votações e da fila global do grupo e que possuem o seguinte formato de *dados*:

- < ***vote***, *id_candidato*, *valor_votacao* >
- < ***message***, *id_msg*, *msg* >

A identificação de **vote**, indica que se trata de um tuplo do tipo votação (*TupleVoteType*) e está associado a um processo de votação. No caso da identificação **message**, trata-se de um tuplo (*TupleMsgType*) com informação referente a uma mensagem da fila global de mensagens do grupo. Estes tipos de tuplos são geridos internamente, dado que se prendem com funcionalidades internas à arquitectura.

No caso dos tuplos, do espaço partilhado oferecido pelo modelo MAGO, o campo *dados* possui o seguinte formato:

< ***data***, *dados_produzidos* >

Onde **data** é um identificador que caracteriza este tuplo com sendo do tipo (*TupleDataType*) que é manipulado ao nível do espaço partilhado do grupo oferecido pelo modelo. Este tipo de tuplo é dependente da informação e da estrutura que a aplicação pretende definir para os dados partilhados do grupo, sendo os dados contidos nestes tuplo produzidos por membros do grupo. Note-se que, é da responsabilidade da aplicação efectuar a inserção e leitura da informação no SI, recorrendo para tal ao módulo de suporte de acesso ao SI[17], que está directamente relacionado com as definições da aplicação. O campo *dados_produzidos*, pode conter a informação referente à forma de acesso ao dados no SI e a sua caracterização, e não os dados propriamente ditos.

- **Actualização de dados no espaço de tuplos -** $update()$
  Esta primitiva coloca informação, sob a forma de um tuplo do tipo *data*, no espaço partilhado do grupo, sendo da sua responsabilidade e com base nos argumentos que lhe são passados, construir e colocar o tuplo no espaço do grupo, assim como definir a forma como os dados são divulgados pelos membros.

  Os seus parâmetros são:

[17] Esse módulo está descrito mais à frente na secção 5.7.

- identificador da entidade produtora;
- nome dos dados no espaço;
- lista de parâmetros definidos pela aplicação (dados);
- tipo de acesso, que indica se os dados são públicos (PUBLIC acessíveis por todos) ou privados (PRIVATE apenas acessíveis pelo seu produtor);
- tipo de notificação, indica a forma como os membros são notificados da existência de novos dados inseridos no espaço partilhado do grupo

O objecto "lista de parâmetros", definido pela aplicação, pode ser, por exemplo: tipo de ficheiro, referência para dados no sistema de informação, etiquetas (*tags*) que caracterizam a informação[18], data da criação dos dados ou outros. O nome, atribuído aos dados, permite efectuar a leitura, ou seja, aceder directamente a um dado tuplo.

A sequência de acções desencadeada pela invocação desta primitiva é a seguinte:

- construção do tuplo;
- colocação do tuplo no espaço, através da primitiva de escrita disponibilizada pelo JGroupSpace;
- se tipo de notificação = NOTIFY_UPD
  - publicação de um evento do tipo UPDATE_EV, com informação relativa ao tuplo que é colocado no espaço (*nome_do_tuplo*), para todos os membros do grupo;
- se tipo de notificação = NO_NOTIFY_UPD
  - não faz mais nada.

- **Leitura não destrutiva de conjunto de tuplos do espaço -** $find()$
  Esta primitiva permite efectuar uma leitura não destrutiva de tuplos do espaço, que obedecem ao padrão de pesquisa especificado. A sua invocação desencadeia um processo de pesquisa sobre o espaço partilhado do grupo, sendo actualizada uma estrutura interna, da entidade, com as cópias dos tuplos de dados, que verificam as condições indicadas. No caso de não existirem tuplos que verifiquem esse padrão de características, é devolvida a lista vazia.

  Tal como na primitiva de $update()$, a partir dos argumentos, é construído um tuplo de pesquisa, com base no qual é desencadeada a leitura dos tuplos do espaço cujo padrão de valores seja compatível[19] com os valores indicados no tuplo de pesquisa. A sequência de acções desencadeada pela invocação desta primitiva é a seguinte:

[18] O conjunto de etiquetas (*tags)*, podem atribuir uma caracterização semântica ao conteúdo dos dados.
[19] Por "pattern matching".

  - construção do tuplo de pesquisa;
  - leitura de tuplos do espaço, através da primitiva de leitura do JGroupSpace (*rdpall*()[20]) que permite obter todos os tuplos do espaço que satisfaçam as condições, devolvendo-os sob a forma de um vector;
  - colocação da informação retornada, numa estrutura interna da entidade, que desencadeou a pesquisa, que pode ser acedida através de uma das primitivas de leitura do modelo (*consult*()).

- **Leitura não destrutiva de tuplo -** *consult*()
  Existem duas primitivas de consulta, uma que efectua a consulta de um tuplo directamente do espaço partilhado e outra que efectua a consulta de um tuplo especifico da estrutura interna que foi actualizada após a execução da primitiva de pesquisa (*find*). As duas primitivas distinguem-se entre si pelos parâmetros especificados quando da sua invocação.

  No primeiro tipo de consulta, é efectuada a leitura, não destrutiva, de um tuplo do espaço partilhado. Após a invocação desta primitiva, é devolvido um tuplo do espaço, que obedeça ao padrão indicado. Tal como para a primitiva de *find*(), é construído um tuplo de pesquisa com base nos argumentos de chamada da primitiva. A leitura do tuplo pode ser: bloqueante (BLOCK), que fica a aguardar que exista um tuplo que obedeça às condições, ou pode ser não bloqueante (NO_BLOCK), situação em que, caso não exista um tuplo no espaço que verifique as condições, retorna um tuplo "vazio". Em geral não é garantido que o tuplo retornado seja o único no espaço que satisfaça as condições; só é garantido que esse tuplo é único, se for passado o parâmetro de entrada com identificador do tuplo (*nome_do_tuplo*), que identifica de forma única um tuplo no espaço partilhado.

  No segundo tipo de consulta, a leitura do tuplo é efectuada sobre a estrutura interna da entidade, na qual se encontra a informação dos tuplos retornados quando da última pesquisa. Neste tipo de consulta, a leitura do tuplo da estrutura tem por base a posição que ele ocupa na estrutura que foi construída após a execução de *find*().

- **Leitura com remoção de tuplo -** *get*()
  Tal como no caso da leitura não destrutiva, foram definidas duas primitivas distintas de *get*.

  Um das primitivas trabalha directamente sobre o espaço de tuplos, efectuando a remoção de um tuplo que satisfaça as condições. No caso de se optar por uma leitura com semântica não bloqueante (NO_BLOCK), se não existir nenhum tuplo que satisfaça as condições retorna de imediato (com tuplo "vazio"). Se existir

[20] Este método possui uma semântica não bloqueante; no caso em que não existam tuplos no espaço que satisfaçam o padrão de pesquisa, devolve vector vazio.

um tuplo, retira um qualquer deles[21]. Na semântica bloqueante (BLOCK), fica a aguardar que exista no espaço um tuplo que satisfaça as condições, para o remover. Esta primitiva é idêntica à primitiva de *consult*(), com a diferença de que um tuplo é removido do espaço partilhado.

Na outra primitiva *get*(), a leitura com remoção é efectuada sobre a estrutura interna que foi actualizada após a execução da primitiva de pesquisa (*find*), ou seja o tuplo é removido da estrutura interna e não do espaço de tuplos.

A manutenção da consistência entre o sistema e informação e o espaço partilhado é da responsabilidade da aplicação e da forma como esta procede à utilização das primitivas. Ou seja, se remover um tuplo do espaço partilhado, também deve proceder à remoção no sistema de informação, dos dados referenciados pelo tuplo (se for esse o caso).

5.7 Suporte da interacção com o sistema de informação

O modelo MAGO reflecte o carácter distribuído das aplicações e o facto de os dados relativos aos grupos e às entidades que os constituem serem armazenados localmente pelos seus membros, isto é os dados são mantidos nos dispositivos computacionais através dos quais os utilizadores interagem. No entanto, revelou-se ser importante a manutenção, em paralelo, de um repositório de dados onde fosse possível manter, de forma aproximada, informação sobre o estado e a evolução de uma aplicação desenvolvida sobre este modelo.

A existência deste Sistema de Informação (SI) permite também garantir a persistência da informação referente à organização do sistema, aos dados próprios e aos conteúdos manipulados. Por exemplo, quando ocorrem situações em que um mesmo utilizador pode efectuar registos no sistema a partir de diferentes dispositivos, quando ocorre uma avaria ou desaparecimento de um dispositivo, é possível recuperar a informação de forma relativamente simples e transparente por parte do utilizador, recorrendo à informação mantida no SI. Isto reveste-se de especial importância, principalmente nos cenários de aplicação, como por exemplo os apresentados anteriormente na secção 4.7 onde a dinâmica de utilização leva a que existam necessidades de gestão e manutenção dos dados gerados pela aplicação, bem com da informação referente à evolução de estado do próprio sistema.

O suporte do sistema de informação serve de base à estruturação e definição das funcionalidades que permitem armazenar e gerir os dados do sistema associados às entidades elementares e aos grupos ao nível de um SI. O suporte oferece uma interface a partir da qual a aplicação pode aceder à informação armazenada. Independentemente

[21]Na sua implementação, utiliza o método *inp*() do JGroupSpace, que é também não bloqueante, sendo o tuplo retornado um qualquer dos que satisfaçam as condições.

da tecnologia que serve de suporte ao SI, foram identificadas as estruturas e interfaces de funções que permitem efectuar a manipulação dos dados. Deve ser oferecida, pelas plataformas de SI, a implementação do conjunto de estruturas e funcionalidade que são aqui descritos. A aplicação tem a responsabilidade de escolher qual a plataforma de suporte que melhor se adapta às suas necessidades e características, devendo recorrer às funcionalidades por estas disponibilizadas para interagir com o SI.

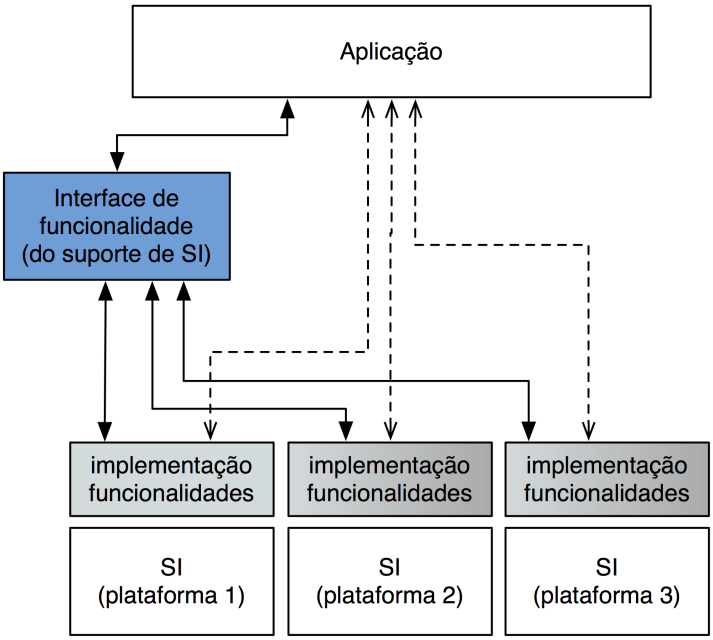

Figura 5.14: Interacção entre a aplicação e o sistema de informação.

Como se pode observar pela figura 5.14, a aplicação recorre ao suporte do sistema de informação para manipular a informação, podendo no entanto interagir directamente com este para operações em que não utilize as funcionalidades da arquitectura do modelo proposto. A interacção com o suporte de SI é efectuada através de uma interface onde é disponibilizado um conjunto de funcionalidades que permitem à aplicação gerir a informação associada aos seus elementos (entidades elementares e grupos e respectivos atributos e conteúdos).

5.7.1 Estruturas de dados de suporte

As estruturas de dados aqui referidas, caracterizam o formato das estruturas que armazenam, no sistema de informação, os dados referentes às entidades elementares e grupos e seus respectivos conteúdos.

Ao nível dos elementos que possuem informação associada, existem as entidades elementares e os grupos, possuindo cada um deles um conjunto de atributos com informação própria do sistema e com conteúdos a estes associados, sendo a manipulação destes dependente da aplicação. Como forma de gerir a informação associada aos vá-

rios elementos presentes no sistema, foram definidas estruturas auxiliares para suportar os dados referentes aos atributos e respectivos valores, assim como para a gestão directa dos membros dos grupos.

Em particular, esta informação vai ser também utilizada como forma de gerir a formação dos grupos implícitos, ou seja o suporte de grupos implícitos recorre também às funcionalidades disponibilizadas para a interacção com o SI.

Entidades Elementares

As entidades elementares, tipicamente associadas a um utilizador registado no sistema, possuem um conjunto de atributos relacionados com a informação do sistema e ainda com informação característica e dependem da aplicação.

Informação própria da entidade:

- identificador único de utilizador (*login*);
- nome do utilizador;
- palavra chave de acesso;
- data de registo no sistema.

A informação dependente da aplicação é composta por um conjunto de pares do tipo: <nome_atributo, valor_atributo>. Os atributos e os valores que estes podem assumir são definidos pela aplicação em estruturas com a informação dos nomes dos atributos e do conjunto de valores que cada um pode tomar. Não é obrigatório que cada atributo tenha associado um conjunto de valores pré-definido, podendo ser definido o seu tipo e deixado o valor em aberto.

Cada entidade, para além da informação denominada própria[22], possui também informação relativa ao conjunto de atributos que a aplicação definiu. Os atributos, como já foi referido, são caracterizados por um nome, um tipo e um conjunto de valores que pode tomar, sendo este último opcional.

Grupos

Para os grupos foram definidas duas estruturas distintas, uma para suporte aos grupos explícitos e outra para os grupos implícitos.

A informação associada aos grupos explícitos é constituída por:

- identificador do grupo (id_grupo);
- identificador do criador do grupo;
- tipo de acesso (política de filiação);

[22] Esta informação é comum a todas as aplicações, independentemente do seu domínio.

- número máximo de membros permitidos no grupo;
- número de membros filiados;
- lista de membros;
- data de criação.

No caso dos grupos implícitos, para além da informação base, é também definido o conjunto de atributos/valores que levaram à sua formação. É com base nesta informação, que caracteriza os diferentes grupos implícitos criados e na informação referente aos atributos das entidade registadas, que é efectuada a gestão dos membros do grupo, ou seja a sua constituição. Deste modo, a informação associada a um grupo implícito é constituída por:

- identificador único do grupo;
- identificador do criador do grupo;
- política de filiação;
- número máximo de membros permitidos;
- número de membros filiados;
- lista de membros;
- data de criação;
- conjunto de atributos/valores, que caracterizam o grupo, ou seja que estão na origem da sua criação.

Conteúdos

Os conteúdos aqui referidos, constituem toda a informação produzida e colocada no sistema pelas entidades. O seu tipo e a sua caracterização depende obviamente da aplicação.

No entanto, é assumido que um conteúdo é sempre produzido, ou seja colocado no sistema, por uma entidade elementar ao qual esta atribuiu uma identificação (nome do conteúdo). Deste modo a informação que está directamente associada a uma entidade, pode vir a ser partilhada por esta, através da sua disponibilização/colocação num grupo. A informação referente aos conteúdos possui a seguinte estrutura:

- nome do conteúdo, que é um identificador único do conteúdo;
- identificador do produtor;
- informação;

- data de entrada;
- anotações referentes ao conteúdo armazenado.

Para além do próprio conteúdo, podem ser guardadas anotações (de acordo com parâmetros especificados pela aplicação), que permitem caracterizar a informação. Assim, permite-se, entre outras opções, efectuar consultas da informação armazenada, de uma forma mais eficiente e organizada de acordo com determinados critérios definidos pela aplicação.

Os conteúdos, ao serem colocados no espaço partilhado do grupo, podem ser acedidos pelos diversos membros. No SI, passa a existir uma estrutura com a seguinte informação:

- identificação do grupo;
- nome do conteúdo.

A colocação de dados no espaço partilhado de um grupo, passa primeiro pela criação de um elemento, com a estrutura associada ao conteúdo, pela actualização da informação no espaço partilhado (*update*) e pela associação do conteúdo ao grupo, ao nível do sistema de informação.

5.7.2 Funções da interface com o sistema de informação

Como forma de gerir e manipular as estruturas de dados definidas no sistema de informação, foi definida uma interface que disponibiliza um conjunto de funcionalidades que podem ser utilizadas para a interacção com o sistema de informação. A implementação destas funções é dependente da plataforma que serve de suporte ao SI. Essas funções são utilizadas quer pela aplicação, quer pelo componente do modelo responsável pela gestão dos grupos implícitos.

As funções disponibilizadas permitem efectuar a criação, actualização e consulta dos dados associados às entidades elementares e grupos ao nível do SI. No entanto e motivado pelas necessidades específicas de cada aplicação podem paralelamente, ser desenvolvidas outras funções ao nível das diversas plataformas.

De seguida são apresentadas as funcionalidades básicas que as diversas interfaces com as plataformas deverão implementar e o formato a que devem obedecer.

Inserção e actualização de uma entidade

Para inserir uma nova entidade, ao nível do sistema de informação, foi definida a função $si\_newEntity()$, que deve obedecer à seguinte sintaxe:

si_newEntity($\underline{in} : login, chave\_acesso; \underline{out} : resultado$)

São passados, como parâmetros de entrada, o valor de *login* e uma *chave_acesso* para validação. É retornado um valor com indicação de operação bem sucedida (*resultado*).

Esta função tem a responsabilidade de criar uma nova entrada referente ao registo de uma entidade elementar no sistema de informação com os dados da sua validação de entrada, *login* e *chave* e uma data que corresponda à data da sua criação, que é atribuída pelo próprio SI. Estes elementos definem sempre uma entidade elementar e existem sempre, em qualquer tipo de aplicação. Após a criação desta entrada, há que actualizar o conjunto de atributos definidos e instanciados pela aplicação, através da invocação da função *si_setEntity*().

si_setEntity($\underline{in} : login, [instancia\_atributo]; \underline{out} : resultado$)

Esta função, de acordo com os atributos específicos definidos pela aplicação, efectua a actualização dos valores dos atributos da entidade previamente criada e identificada pelo seu *login*. A lista de instâncias de atributos é constituída por pares de valores do tipo, identificação do atributo e o seu valor, sendo a atribuição do valor efectuada de acordo com o funcionamento específico da aplicação.

A informação referente a uma entidade, ou seja o seu conjunto de atributos, pode ser consultada através da função:

si_consultEntity($\underline{in} : login; \underline{out} : [instancia\_atributo]$)

Esta função retorna o conjunto de atributos/valores da entidade cuja identificação é indicada (*login*).

Remoção de uma entidade

A função de remoção[23] de uma entidade tem como objectivo remover do sistema de informação a informação associada à entidade. No entanto, os dados produzidos pela entidade, mas que foram disponibilizados num grupo, mantém-se acessíveis nesse grupo; somente os seus dados próprios não partilhados são removidos. Os conteúdos mantém-se no sistema enquanto existir uma entidade elementar ou grupo a que estes se encontrem associados.

si_removeEntity($\underline{in} : login; \underline{out} : resultado$)

Esta função retorna um valor com a indicação de operação bem sucedida (*resultado*).

[23] A remoção aqui referida corresponde na realidade a uma desactivação do registo.

Manipulação dos conteúdos de uma entidade

Cada entidade pode ter associados dados por ela produzidos, ou seja pode ter associado um conjunto de conteúdos (textos, imagens, sons ou vídeos) que pode vir a partilhar nos seus grupos ou manter como privados. O sistema de informação tem a responsabilidade de definir o conjunto de estruturas que lhe permitam armazenar esse tipo de informação e relacioná-la directamente com a entidade produtora. Para a actualização de dados próprios da entidade no sistema de informação, deve invocar a função $si\_updateCEntity()$.

si_updateCEntity$(\underline{in} : login, objecto\_dados; \underline{out} : data\_id)$

O $objecto\_dados$, passado como argumento é constituído pelo conteúdo e pela informação a este associada e já anteriormente referida[24]. A configuração, o tratamento dessa informação e a manipulação dos conteúdos é da responsabilidade da aplicação. É retornado um identificador (único) do conteúdo $data\_id$, que permite aceder directamente aos dados.

Para a obtenção de um conteúdo produzido por uma dada entidade, deve ser utilizada a função $si\_getCEntity()$.

si_getCEntity$(\underline{in} : login, data\_id; \underline{out} : objecto\_dados)$

São indicados como argumentos, o identificador da entidade ($login$) e o identificador do conteúdo a que se pretende aceder ($data\_id$). O identificador dos dados é um valor retornado quando da actualização dos dados.

A função $si\_removeCEntity()$ tem como objectivo efectuar a remoção de conteúdos associados a uma dada entidade.

si_removeCEntity$(\underline{in} : login, [data\_id]; \underline{out} : resultado)$

São removidos do sistema de informação todos os conteúdos cujos respectivos identificadores sejam indicados, como argumentos ($[data\_id]$)[25]. No caso de não ser indicado nenhum identificador de conteúdos, são removidos todos os conteúdos privados, ou seja, não partilhados, associados à entidade.

A função $si\_consultCEntity()$ permite obter todo o conjunto de identificadores de conteúdos de uma dada entidade:

si_consultCEntity$(\underline{in} : login; \underline{out} : [data\_id])$

[24] Ver a descrição da estrutura de suporte de conteúdos.

[25] Mais uma vez a remoção aqui referida corresponde na realidade a uma desactivação.

Inserção e actualização de um grupo

Tal como para as entidades, também os grupos possuem um conjunto de funções que permitem efectuar a manipulação dos seus dados e estrutura, ao nível do sistema de informação.

A criação de uma entrada no SI para um novo grupo é efectuada pela função *si_newGroup*().

si_newGroup($\underline{in}$: *nome_grupo*, *login*; $\underline{out}$: *resultado*)

Nesta função é indicado como parâmetro de entrada o *nome_do_grupo*, sendo também indicado o identificador da entidade (*login*) responsável pela criação do novo grupo. Esta função retorna um valor (*resultado*), com a indicação de operação bem sucedida.

Após a criação desta entrada, deve ser efectuada a actualização e atribuição do conjunto de parâmetros que definem o grupo criado tais como a política de filiação e o número máximo de membros admitidos. Esta actualização dos dados do grupo, ao nível do sistema de informação, é efectuada pela função:

si_setGroup($\underline{in}$: *id_grupo*, *nome_grupo*, *lista_param*, [*atributos_implicitos*]; $\underline{out}$: *resultado*)

No caso dos grupos implícitos, para além da lista de parâmetros *lista_param*, é também passado o conjunto de atributos e respectivos valores, que estão na origem da criação do grupo. O parâmetro de entrada *id_grupo* é o valor retornado como identificador do grupo pela primitiva de criação de grupo *create*().

A informação referente a um grupo, ou seja o seu conjunto de parâmetros e de atributos, pode ser consultado através da função:

si_consultGroup($\underline{in}$: *id_group*; $\underline{out}$: *lista_param*, [*atributos_implicitos*])

Esta função retorna toda a informação própria associada ao grupo. No caso de se tratar de um grupo implícito, para além das características próprias do grupo, é retornada a lista de pares atributo/valor que foram definidos para a criação do grupo.

Remoção de grupo

A função de remoção de um grupo do SI tem como objectivo remover a informação associada ao grupo[26]. São também removidos os conteúdos partilhados no grupo e

[26]Também aqui a remoção corresponde a uma desactivação da entrada do grupo, podendo os dados ser utilizados na construção de um histórico.

que foram colocados por entidades que já não existem no sistema. No caso dos conteúdos estarem associados a entidades que se encontram ainda no sistema, esses não são removidos, sendo removida somente a sua associação ao grupo.

si_removeGroup($\underline{in}$: id_grupo; $\underline{out}$: $resultado$)

Gestão de membros de um grupo

A informação referente à constituição do grupo ao nível do SI, é actualizada através das funções $si\_putGroupMember()$ e $si\_removeGroupMember()$. A primeira associa uma entidade a um grupo e a segunda retira uma dada entidade de um grupo.

si_putGroupMember($\underline{in}$: $id\_grupo, login$; $\underline{out}$: $resultado$)
si_removeGroupMember($\underline{in}$: $id\_grupo, login$; $\underline{out}$: $resultado$)

De notar que com a remoção *si_removeGroupMember* , somente a entidade é removida do grupo, devendo ser mantidos no grupo os conteúdos disponibilizados por esta.

Manipulação dos conteúdos de um grupo

Ao nível da aplicação, cada grupo pode ter associado um conjunto de conteúdos (textos, imagens, sons ou vídeos), que são colocados no grupo pelos seus membros. Tal como para os conteúdos pertencentes às entidades elementares, o sistema de informação deve ter definidas as estruturas que lhe possibilitem efectuar a gestão dos conteúdos do grupo.

Para a atribuição/actualização de um conteúdo num grupo, por parte de uma entidade membro, foi definida a função $si\_updateCGroup()$.

si_updateCGroup($\underline{in}$: $id\_grupo, login, objecto\_dados$; $\underline{out}$: $data\_id$)

O parâmetro *login* identifica a entidade produtora do conteúdo. O valor retornado *data_id* possibilita a identificação e o posterior acesso ao conteúdo.

Para aceder a um conteúdo pertencente a um dado grupo, deve ser utilizada a função $si\_getCGroup()$.

si_getCGroup($\underline{in}$: $id\_grupo, data\_id$; $\underline{out}$: $objecto\_dados$)

Na chamada desta função, deve ser indicado o identificador que foi atribuído ao conteúdo a que se pretende aceder, sendo este valor atribuído quando da actualização dos dados (*si_updateCGroup*).

A função *si_removeCGroup*() tem como objectivo efectuar a remoção de conteúdos que são pertença de um grupo[27].

si_removeCGroup($\underline{in}$: $id\_grupo, [data\_id]; \underline{out}$: $resultado$)

São removidos do grupo todos os conteúdos cujos identificadores sejam indicados, como argumentos ([*data_id*]).

No caso de, em vez de ser passada uma lista de identificadores de conteúdos, ser indicado o identificador de uma entidade (*login*), serão removidos do grupo todos os conteúdos produzidos/partilhados por essa entidade no grupo. O conteúdo só será no entanto removido do sistema de informação, se não se encontrar associado a nenhuma outra entidade e/ou grupo, caso contrário será removida somente a associação do conteúdo ao grupo.

A função *si_consultCGroup*() permite obter todo o conjunto de identificadores de conteúdos de um dado grupo, produzidos por uma dada entidade (*login*):

si_consultCGroup($\underline{in}$: $id\_grupo, [login]; \underline{out}$: $data\_id$)

No caso de não ser indicado um valor de *login*, serão retornados os identificadores de todos os conteúdos partilhados no grupo pelos seus membros.

5.8 Gestão de grupos implícitos

Como já foi referido, os grupos implícitos possuem características que os distinguem dos grupos explícitos, sendo as principais diferenças entre os dois, a forma como é desencadeado o seu processo de criação e a forma como é gerido o processo de filiação dos membros.

Contrariamente aos grupos explícitos, definidos de forma explicitamente programada pelos seus membros, os grupos implícitos são gerados sem intervenção explícita por parte do programa da entidade (utilizador). A filiação num grupo implícito emerge naturalmente das características individuais do utilizador, que são estabelecidas ao nível da aplicação. Estas características levam as entidades elementares a serem inseridas em grupos com quem eventualmente não têm qualquer tipo prévio de contacto ou que nem mesmo conhecem.

A gestão dos grupos implícitos está directamente dependente do sistema de informação, o que não se verificava nos grupos explícitos. Nos grupos implícitos toda a gestão de membros, tem por base os dados dos diversos membros do sistema, existentes no SI. Os critérios de selecção dos membros de um dado grupo implícito são definidos pela aplicação e passados como parâmetros, quando da invocação da primitiva

[27] Os conteúdos do grupo não são propriamente removidos, são desactivados, de modo também a permitir a gestão do histórico.

create(). É da responsabilidade da aplicação a definição das propriedades e campos que caracterizam cada entidade elementar do sistema, assim como a sua gestão.

Como já foi referido, o processo de criação e de gestão de filiação dos grupos implícitos é distinto do dos grupos explícitos, sendo de seguida apresentada uma descrição da forma como foram implementadas as primitivas para a criação e filiação e a forma como se integram com o suporte do sistema de informação.

O processo de gestão de filiações nos grupos implícitos é desencadeado por duas situações distintas quando da criação do grupo e quando se procede a uma modificação das características/atributos de uma dada entidade.

O processo que é desencadeado em cada uma dessas situações é descrito nas duas subsecções seguintes.

5.8.1 Criação de grupo implícito

A invocação da primitiva *create*() desencadeia o processo de criação do grupo e da filiação das entidades registadas que satisfaçam o conjunto de características a que os membros do novo grupo devam obedecer.

Na sequência do processo de criação, é desencadeada uma pesquisa sobre a informação referente às entidades actualmente registadas no sistema e que se encontra no sistema de informação.

A definição do processo de pesquisa, com base nos parâmetros indicados pela entidade criadora do grupo, é da responsabilidade do programador da aplicação. Isto porque o processo de pesquisa se encontra directamente relacionado com o sistema de informação que serve de suporte à aplicação, devendo no entanto obedecer à seguinte interface:

$$\textbf{findMembers}(\underline{in} : lista\_param; \underline{out} : [lista\_membros], numMembros)$$

Em que a *lista_param* é uma lista de elementos do tipo: <atributo,valor>, que caracterizam as condições de entrada no grupo. É retornada uma lista com os identificadores das entidades, actualmente registadas no sistema, que satisfaçam todas as condições de criação do grupo. É também retornado o *numMembros* que indica qual o número de elementos que inicialmente fazem parte do grupo (ou seja o número de elementos da *lista_membros*).

A sequência de criação de um grupo implícito, segue os seguintes passos (ver figura 5.15):

- a entidade que pretende criar o grupo, comunica com o servidor de grupo universal para a criação de um grupo implícito de acordo com os parâmetros;
- o servidor de grupo universal comunica com o servidor de grupos, com vista à criação, por parte deste, de um novo grupo e ao lançamento do seu representante;

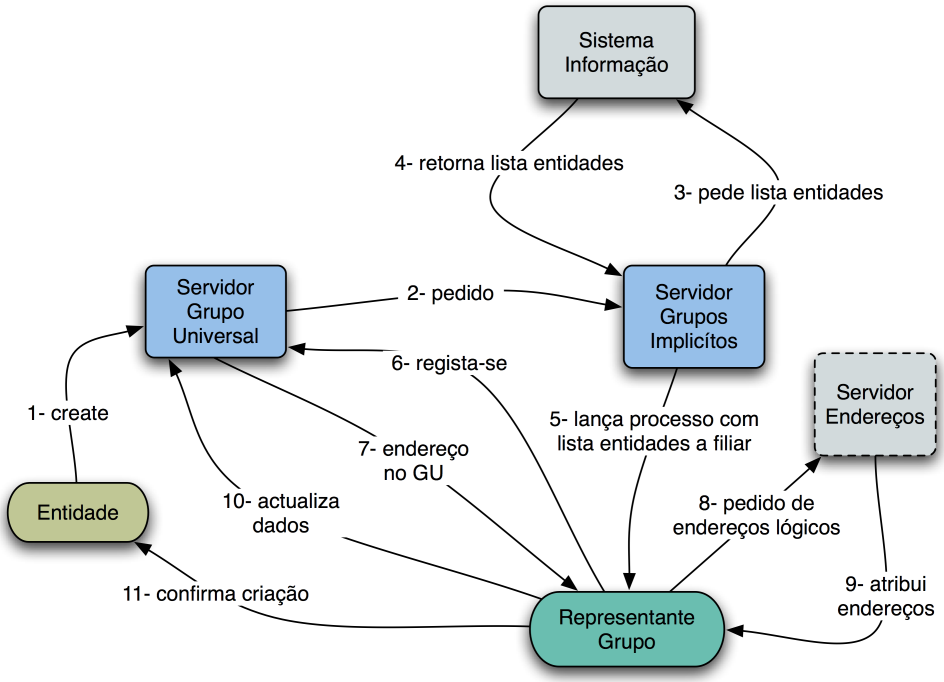

Figura 5.15: Processo de criação de um grupo implícito.

- o servidor de grupos, ao receber o pedido, deve:
  - validar o nome do grupo, de forma a garantir que este é único;
  - comunicar com o servidor de grupos implícitos, passando a lista com os parâmetros para a pesquisa de membros;
  - aguardar o resultado da pesquisa, ou seja a lista de entidades que satisfazem os critérios especificados na criação do grupo;
  - receber a lista de candidatos a membro e lançar o representante do grupo implícito, passando-lhe a lista de entidades que foram encontradas;
- o representante de grupo, para além das tarefas já descritas para os grupos explícitos (ver secção 5.5.3), deve:
  - comunicar com cada uma das entidades (pertencentes à lista) através da invocação de um método específico definido na interface das entidades elementares. Este método é o responsável por gerir a filiação das entidades nos grupos implícitos, ou seja cria a instância do grupo, procede à sua filiação (*join*) e actualiza a informação no SI, através de um processo semelhante ao desencadeado na filiação em grupos explícitos;
  - comunicar com a entidade criadora confirmando a criação do grupo, após todas as filiações terem sido processadas.

O servidor de grupos implícitos é o elemento responsável por gerir o processo de pesquisa de membros para os grupos implícitos. Essa pesquisa tem por base uma função de pesquisa, que é suportada ao nível da implementação das funcionalidades de acesso ao SI.

5.8.2 Actualização de dados

As alterações na constituição dos grupos implícitos dependem do facto de as entidades poderem modificarem o seu conjunto de atributos/características[28].

A aplicação deve garantir que, após a execução de uma operação de actualização de dados (atributos) de uma entidade no sistema de informação, se procede à invocação de uma função $updateProfile()$. Essa função desencadeia o conjunto de acções que levam à actualização da constituição dos diversos grupos implícitos existentes.

updateProfile($\underline{in} : entidade\_id, [lista\_novos\_parametros]; \underline{out} : resultado$)

A execução desta função de $updateProfile()$, ao nível da entidade que sofreu as alterações, desencadeia-se em dois passos:

1º passo: Invocar o método remoto ($updateImplicits()$) do servidor de grupos implícitos:

updateImplicits($\underline{in} : id, [lista\_param], \underline{out} : [grupos\_remover], [grupos\_inserir]$)

Este método procede à seguinte sequência de acções (figura 5.16):

- verifica se a entidade pertence a algum dos grupos implícitos existentes:
  - para cada grupo G a que pertence, deve verificar se os novos dados satisfazem as condições de pertença ao grupo:
    - se sim: ok;
    - se não: coloca G na lista de *grupos_remover*;
  - verifica se existe algum grupo implícito G' (daqueles a que não pertence), que satisfaça os novos dados:
    - se sim: coloca G' na lista *grupos_inserir*;
    - se não: ok;
- retorna, para a entidade, lista com identificação dos grupos de onde se deve remover e onde se deve filiar.

[28] Para além da modificação de atributos, também há que considerar obviamente a entrada e saída de entidades no sistema. No entanto, estas duas situações podem ser consideradas como casos particulares de alteração de características.

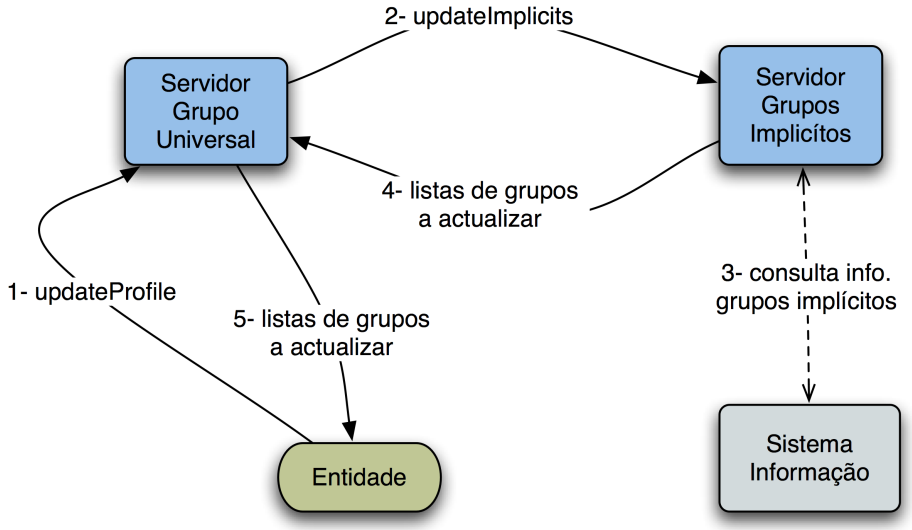

Figura 5.16: Alteração da constituição de um grupo implícito.

2º passo: Proceder à actualização da sua estrutura de grupos, ou seja, após receber as listas a função $updateProfile()$ vai:

- para grupos da lista de remoção: executar primitiva $leave()$ e remover a instância de grupo respectiva;
- para grupos da lista de inserção: criar instância de grupo e executar primitiva $join()$.

A invocação da função de $updateProfile()$ fica a cargo do programador de aplicação, devendo ser executada sempre que é efectuada uma operação de modificação dos dados referentes a uma entidade no sistema de informação.

5.9 Tipos de eventos suportados

Como já foi referido, a realização proposta do modelo MAGO, define um conjunto de eventos, definidos ao nível da própria implementação, que servem de suporte à implementação de algumas das primitivas propostas. Estes eventos, desencadeados por execução de chamadas a algumas das primitivas do modelo, são descritos de seguida, sendo identificados os elementos responsáveis pela sua produção e recepção e qual a sequência de acções que é seguida pela rotina de atendimento nos receptores e que caracteriza o seu comportamento.

A descrição dos eventos, apresentada seguidamente, está estruturada de acordo com a primitiva que desencadeou o evento.

DESTROY_EV

Este evento está associado à primitiva de remoção de um grupo $destroy()$, quando

invocada com a opção de NOTIFICATION, ou seja, quando ao desencadear o processo efectivo de remoção de um grupo do sistema, os seus membros são avisados desse facto.

- produtor do evento: entidade que pretende proceder à remoção;
- receptores do evento: representante do grupo e os seus membros.

A entidade que desencadeia o processo de remoção tem que ser membro do grupo. Ao receber esta notificação, os membros do grupo podem desencadear um processo de encerramento, definido e configurado pela aplicação, para além de procederem ao cancelamento das operações em curso referentes ao grupo e à desactivação da recepção de eventos do mesmo. A aplicação pode estabelecer o comportamento a ser desencadeado, através da definição de uma função que é executada quando da recepção deste evento por parte das entidades. No caso do representante de grupo, ao receber a notificação de encerramento, inibe e encerra operações que se encontrem em curso (como por exemplo processos de votação) e termina a sua execução.

DESTROY_ACK_EV

Este evento está também associado à primitiva de remoção de grupo *destroy*, mas quando invocada com a opção de NOTIFICATION_ACK. Com esta opção, é aguardada uma confirmação de saída por parte dos membros do grupo.

- produtor do evento: entidade que pretende proceder à remoção do grupo;
- receptores do evento: representante do grupo e os seus membros.

Ao receber esta notificação os membros do grupo, para além de procederem ao cancelamento das operações em curso que envolvem o grupo, desencadeiam a sua saída explícita do mesmo através da primitiva *leave*(). No caso do representante de grupo procede, tal como no caso anterior, ao cancelamento das operações em curso sobre o grupo e aguarda pela saída dos membros antes de terminar a sua execução e encerrar o grupo.

JOIN_EV

Este evento está directamente relacionado com a execução da primitiva de filiação *join*(). O pedido de filiação, por parte de uma entidade, é efectuado ao representante do grupo. No caso de se tratar de um pedido de entrada sujeito a votação, desencadeia-se a notificação de votação em curso, aos membros do grupo.

- produtor do evento: representante de grupo, ao receber um pedido de filiação por parte de uma entidade;
- receptores do evento: membros do grupo.

A publicação do evento é acompanhada da informação referente à votação, ou seja o identificador do tuplo que funciona como urna de votos.

Por parte de cada entidade, ao receber esta notificação e caso pretenda votar positivamente a entrada de candidato a membro, desencadeia a seguinte sequência de acções:

- constrói tuplo de votação com base nos dados recebidos e com identificador do tuplo
- efectua leitura bloqueante com remoção do tuplo do espaço partilhado (*in*())
- processa tuplo de votação lido do espaço, se campo de votação deste for:
  - >0, escrita de tuplo com campo de votação actualizado
  - =0, escrita novamente do tuplo no espaço, sem modificação dos seus campos.

Foi configurado um valor para a duração máxima da votação; se este for excedido assume-se que a votação foi cancelada.

SET_MODE_EV

Este tipo de evento é publicado pelas entidades que pretendem alterar o seu estado de actividade e é desencadeado pela invocação da primitiva *set_mode*(). A informação sobre a modificação do estado, por parte da entidade produtora do evento, é tratada pelo representante do grupo, que efectua a alteração de estado da entidade na sua estrutura de grupos.

- produtor do evento: entidade que desencadeou a modificação do seu estado;
- receptores do evento: representante do grupo e os membros.

MESSAGE_EV

Este tipo de evento é publicado pelo representante de grupo, quando da recepção de uma mensagem dirigida ao grupo com a opção de NOTIFY, situação em que os membros do grupo devem ser informados da chegada de mensagem.

- produtor do evento: representante de grupo, ao receber uma mensagem com a opção de NOTIFY;
- receptores do evento: membros do grupo.

Ao receber esta notificação, cada entidade deve desencadear o processamento de entrega de mensagem, definido pela aplicação. Tal como já foi referido para outros tipos de eventos, é da responsabilidade da aplicação estabelecer o comportamento a

ser desencadeado através da função definida para o efeito, ou seja a forma como vai ser efectuada a leitura da mensagem. Os receptores recebem, juntamente com a notificação, a informação da identificação da mensagem que se encontra para leitura na fila global do grupo.

MESSAGE_SPREAD_EV

Este tipo de evento é publicado pelo representante de grupo, quando da recepção de mensagem para ser entregue aos membros do grupo com opção de SPREAD. Ao receber esta notificação, o representante desencadeia a entrega da mensagem através de um processo de notificação onde a própria mensagem é transmitida aos membros do grupo.

- produtor do evento: representante de grupo, ao receber uma mensagem com a opção de SPREAD;
- receptores do evento: membros do grupo.

ADVERTISE_EV

Este evento é publicado por um membro do grupo que pretende definir um novo tipo de evento no espaço do grupo e é desencadeado quando da invocação da primitiva de *advertise*() com opção de PASSIVE.

O representante do grupo, ao receber este tipo de evento, desencadeia uma rotina que actualiza a estrutura de eventos do grupo que é gerida por ele. Nesta estrutura fica a informação referente ao evento, como por exemplo a referência ao método de atendimento que foi definido, pela aplicação, para o evento.

- produtor do evento: entidade produtora/anunciante do novo tipo de evento;
- receptores do evento: representante do grupo.

Note-se que neste tipo de anúncio, somente o representante de grupo é notificado, sendo da responsabilidade de cada membro do grupo desencadear o processo de consulta dos tipos de eventos disponíveis no grupo (*chk_advertise*()), para detectar quais os tipos de eventos disponíveis no grupo e que podem ser subscritos.

ADVERTISE_ACT_EV

Este evento é publicado também pelo membro do grupo que pretende definir um novo tipo de evento no grupo e é também desencadeado pela invocação da primitiva *advertise*(), mas neste caso com a opção de ACTIVE. Nesta situação todos os membros do grupo recebem a notificação de anúncio de um novo tipo de evento e não somente o seu representante, como acontecia com a opção de PASSIVE.

Ao receber esta notificação, os membros do grupo, podem optar por subscrever o novo tipo de evento, através da invocação da primitiva de *subscribe*().

- produtor do evento: entidade produtora, anunciante do novo tipo de evento;
- receptores do evento: membros e representante do grupo.

O método de atendimento[29] que deve ser executado quando da recepção do novo tipo de evento, é transmitido quando da notificação de anúncio. Esta informação deve ser guardada pelos membros do grupo e pelo seu representante na sua estrutura de suporte à gestão de eventos.

UNADVERTISE_EV

Este evento é publicado pelo membro do grupo que pretende remover um tipo de evento no espaço do grupo.

Ao receber este tipo de evento, o representante do grupo desencadeia a rotina de actualização da estrutura de eventos do grupo por ele gerida.

- produtor do evento: entidade produtora e responsável pelo anúncio do tipo de evento, que pretende desactivar;
- receptores do evento: representante do grupo.

Note-se que somente o representante é notificado da desactivação do tipo de evento, no entanto os restantes membros podem proceder em qualquer instante à sua desactivação (*unadvertise*())por consulta dos tipos de eventos suportados pelo grupo (*chk_advertise*()), cuja informação é mantida actualizada pelo representante.

SUBSCRIBE_EV

Este evento é desencadeado pelas entidades que pretendem subscrever um dado tipo de evento que se encontra anunciado no grupo, ou seja quando ocorre uma invocação da primitiva de subscrição (*subscribe*()).

- produtor do evento: entidade do grupo que pretende subscrever um dado tipo de evento, previamente anunciado;
- receptores do evento: representante do grupo.

Ao receber a notificação de subscrição, o representante de grupo deve proceder à actualização da sua estrutura de eventos do grupo, onde é mantida a informação referente aos tipos de eventos suportados no grupo e aos membros que os subscreveram. Ao desencadear este evento, a entidade envia também a informação referente à sua identificação, de forma a poder ser contactada directamente pelo representante de grupo para lhe transmitir a informação referente ao tipo de evento que subscreveu. Essa informação é composta pela referência ao método de atendimento que deve ser executado, pela entidade, quando da posterior recepção deste tipo de evento. O método de atendimento aqui referido é definido ao nível da aplicação.

[29] A definição do método de novos tipos de eventos é da responsabilidade da aplicação.

UNSUBSCRIBE_EV

Este evento é desencadeado pelas entidades que pretendem cancelar a subscrição de um dado tipo de evento que se encontra anunciado no grupo, através da primitiva de *unsubscribe*().

- produtor do evento: entidade do grupo que pretende desactivar a recepção de um dado tipo de evento, previamente subscrito;
- receptores do evento: representante do grupo.

Ao receber a notificação deste tipo de evento, o representante de grupo deve proceder à actualização da sua estrutura de eventos do grupo, onde é mantida a informação referente aos tipos de eventos suportados no grupo e aos membros que os subscreveram.

PUBLISH_EV

Este evento é desencadeado pelas entidades que anunciaram um dado tipo de evento, quando da invocação da primitiva *publish*().

- produtor do evento: entidade responsável pela produção do tipo de evento;
- receptores do evento: membros do grupo.

Ao desencadear este evento, é enviada a informação referente ao tipo de evento que está a ser publicado. Ao receber este evento, as entidades que subscreveram o tipo de evento indicado, desencadeiam o método de atendimento que foi atribuído e definido pela entidade produtora[30] para o seu tratamento. Este método é conhecido pelos membros do grupo que subscreveram o tipo de evento que ocorreu, ou seja que foi publicado no grupo.

UPDATE_EV

Este tipo de evento é recebido pelos membros quando da invocação da primitiva *update*(), ou seja quando é colocada informação (tuplo) no espaço partilhado com a opção de NOTIFY. É da responsabilidade da aplicação definir o comportamento a seguir quando da ocorrência deste tipo de evento, ou seja, a programação deste método de atendimento, fica a cargo do programador da aplicação.

- produtor do evento: membro do grupo responsável pela colocação da informação no espaço partilhado, ou seja entidade que executou a primitiva *updade*();
- receptores do evento: membros do grupo.

[30] Definido ao nível da aplicação.

Os membros do grupo, ao receberem esta notificação são informados de que ocorreu uma modificação do espaço partilhado do grupo, sendo da responsabilidade da aplicação definir o comportamento a ser desencadeado posteriormente.

Evento	Produtor	Receptor	Desencadeado por
DESTROY_EV	entidade membro	representante grupo	primitiva *destroy()*
DESTROY_ACK_EV	entidade membro	representante e membros do grupo	primitiva *destroy()* c/NOTIFICATION_ACK
JOIN_EV	representante grupo	membros grupo	primitiva join()
SET_MODE_EV	entidade	representante e membros do grupo	primitiva *set_mode()*
MESSAGE_EV	representante grupo	membros grupo	recepção mensagem pelo grupo c/NOTIFY
MESSAGE_SPREAD_EV	representante grupo	membros grupo	recepção mensagem pelo grupo c/SPREAD
ADVERTISE_EV	entidade anunciante	representante grupo	primitiva *advertise()* c/PASS
ADVERTISE_ACT_EV	entidade anunciante	representante grupo	primitiva *advertise()* c/ACT
UNADVERTISE_EV	entidade anunciante	representante grupo	primitiva *unadvertise()*
SUBSCRIBE_EV	entidade subscritora	representante grupo	primitiva *subscribe()*
UNSUBSCRIBE_EV	entidade subscritora	representante grupo	primitiva *unsubscribe()*
PUBLISH_EV	entidade produtora	membros grupo	primitiva *publish()*
UPDATE_EV	entidade membro	membros grupo	primitiva *update()*

Tabela 5.1: Tabela de eventos suportados.

Na tabela 5.1, encontra-se o resumo dos tipos de eventos suportados ao nível da implementação da arquitectura.

5.10 Conclusão

Ao longo deste capítulo descreveram-se as opções tomadas na implementação das várias primitivas propostas no modelo e dos vários módulos que permitem a criação de uma arquitectura de um sistema de suporte ao desenvolvimento de aplicações.

Foram descritos os vários componentes constituintes da arquitectura de suporte ao modelo, ao nível da sua estrutura, comportamento e forma de interacção com os restantes elementos.

Foi também descrita a funcionalidade da interface que deve ser observada pelas possíveis plataformas de suporte ao sistema de informação e o tipo de dados e estruturas, directamente relacionados com o modelo, que estas devem suportar. Esta

descrição reveste-se de especial importância, devido principalmente à sua interligação directa com a forma como é suportada a gestão dos grupos implícitos pelo modelo.

Tendo por base a implementação da arquitectura de suporte ao modelo, que foi aqui apresentada, no capítulo seguinte, discute-se a sua utilização/integração para o desenvolvimento de aplicações.

6

Desenvolvimento de aplicações

Conteúdo

Neste capítulo discute-se a metodologia para o desenvolvimento de aplicações com base no modelo MAGO. São apresentadas as necessidades e requisitos genéricos, assim como as configurações de sistema, que têm que ser observadas por qualquer aplicação baseada no modelo. São revistas as funcionalidades típicas que podem ser utilizadas pelas aplicações tendo por base o modelo. É apresentado um protótipo onde foram testados os diversos elementos necessários para o desenvolvimento de aplicações interactivas, em particular os acessos ao sistema de informação e a integração dos restantes módulos presentes na arquitectura definida no capítulo 5. São também apresentadas e discutidas formas de integração em aplicações já existentes com o objectivo de simplificar a definição das interacções e funcionalidades disponibilizadas por estas.

6.1 Introdução

Ao longo deste capítulo são apresentadas formas de utilização da arquitectura do modelo proposto em aplicações com necessidades específicas, onde é necessária a formação de estruturas de grupos, de acordo com as suas características e especificações particulares. Embora as aplicações possam surgir em diferentes ambientes de utilização, possuem um mesmo subconjunto de características e necessidades comuns, obedecendo a idênticos tipos de requisitos.

Nesta secção, revêem-se alguns dos factores envolvidos no desenvolvimento de aplicações em ambientes interactivos distribuídos, tendo como base a implementação do modelo que foi anteriormente apresentada (capítulo 5). A proposta de modelo apresentada tem como principal objectivo simplificar a utilização dos mecanismos de comunicação e de partilha numa estrutura de grupos de utilizadores, oferecendo funcionalidades de gestão de grupos, de interacção entre elementos e de partilha de dados. Estas funcionalidades são disponibilizadas, às aplicações, sob a forma de classes definidas na linguagem Java. Estas classes definem conjuntos de métodos através dos quais se podem modelar e implementar as funcionalidades, que regulam a forma como as entidades do sistema interagem e se relacionam.

As aplicações desenvolvidas, tendo por base o sistema que suporta o modelo MAGO, tiram partido das funcionalidades oferecidas por este, podendo deste modo agilizar toda a fase de desenvolvimento das mesmas. De um modo geral, as aplicações são criadas tendo em vista um conjunto de clientes/utilizadores que interagem com o sistema e entre si através de uma interface gráfica, que lhes dá acesso às funcionalidades e serviços. De um modo geral, o modelo MAGO promove uma organização em que uma aplicação é constituída tipicamente por quatro módulos (ver figura 6.1). Esta organização aplica-se independentemente das funcionalidades pretendidas quer se trate, por exemplo, de jogos, ferramentas de apoio ao ensino ou suportes informativos para espaços culturais. Em todos estes domínios, existe a necessidade de definir:

- a interface, que disponibiliza mecanismos através dos quais os utilizadores interagem com a aplicação;

- o sistema de informação, que serve de suporte para arquivo e acesso os dados referentes à aplicação e ao próprio sistema;

- uma configuração do MAGO, que oferece as funcionalidade, através de um conjunto de primitivas, que permitem à aplicação a caracterizar e definir o tipo de interacções entre os utilizadores e a sua organização em estruturas mais complexas (grupos);

- o motor da aplicação, responsável por definir o comportamento da aplicação, recorrendo às funcionalidades disponibilizadas pelos restantes módulos.

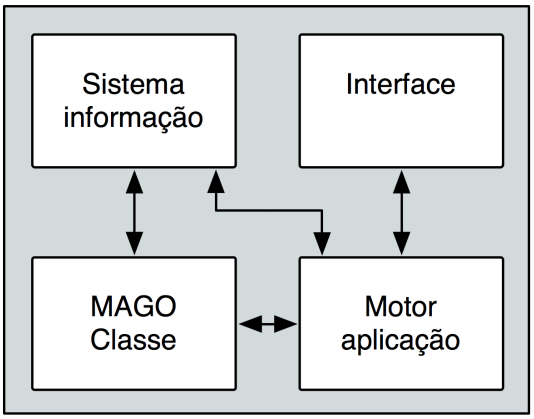

Figura 6.1: Módulos base constituintes de um sistema MAGO para suporte de uma aplicação

6.1.1 Interface

A definição da interface das aplicações reveste-se de especial importância [Nie00], para modelar a interacção entre múltiplos utilizadores e entre estes e o sistema. Nas aplicações consideradas, de um modo geral, a interface do lado do cliente deve ser simples de utilizar e permitir a manipulação simples de conteúdos, bem como possuir um suporte computacional para a execução de processos a nível local.

Na fase de desenvolvimento de aplicações de teste ao modelo foram analisadas diferentes tecnologias de criação de aplicações do lado do cliente, sendo de salientar as baseadas em tecnologias Web, na linguagem Java e ainda suportadas por "Rich Media Applications"[1].

As opções, no que respeita ao suporte do desenvolvimento da interface da aplicação, são praticamente independentes do tipo de funcionalidades oferecidas por esta e prendem-se essencialmente com opções de base tecnológica. Esta escolha tem no entanto consequências ao nível da implementação e estruturação da aplicação e da forma como se efectua a interligação entre os elementos constituintes do sistema e o tipo de dispositivos clientes utilizados, bem como a caracterização do tipo de cenário de aplicação a que se destina.

A arquitectura do modelo foi desenvolvida em linguagem Java, o que torna a sua integração com interfaces desenvolvidas neste mesmo suporte mais simples. No entanto, ao nível do desenvolvimento gráfico e da manipulação de conteúdos

[1]Esta designação pretende englobar um conjunto de soluções tecnológicas onde se enquadram o Adobe Flash [HH02] e/ou Flex.

multimédia, esta opção não é a mais simples, devido à complexidade inerente à criação de interfaces gráficas mais elaboradas em Java.

Para o desenvolvimento de interfaces de aplicações onde existe uma forte interacção por parte do utilizador e uma manipulação de diferentes tipos de conteúdos, existe um conjunto soluções que permitem uma mais fácil manipulação de diferentes tipos de *media*.

Soluções que recorrem a ferramentas de desenvolvimento do tipo das propostas pela Macromedia e posteriormente pela Adobe, permitem a rápida criação de interfaces e possibilitam a manipulação de dados audiovisuais de forma simples, por parte do programador de aplicações. A arquitectura das aplicações desenvolvidas com base nestas ferramentas assenta, de um modo geral, num servidor, que permite efectuar a interligação com a implementação do modelo, sendo a execução, do lado do cliente, vista como uma aplicação independente.

Uma abordagem para o desenvolvimento de aplicações assume que a interface da aplicação do lado do cliente é uma interface Web, o que permite generalizar a forma como é utilizada pelos diferentes dispositivos terminais, constituindo-se, por esse motivo, como a solução adequada quando se pretende que a interacção do utilizador com a aplicação seja efectuada através de *browsers*. Neste tipo de solução, existe um servidor que manipula e gere os objectos Java definidos pelo modelo, sendo as aplicações do lado do cliente, desenvolvidas utilizando a combinação de tecnologias Web como HTML [MK06, FF06], css [Hol03], Javascript [Fla98, HP02] ou Ajax [UD07, Per06].

Esta solução, onde a forma de interacção do utilizador com a aplicação é efectuada através de *Web browser*, coloca alguns problemas relacionados com a integração das tecnologias de desenho de interfaces Web com as de suporte da infraestrutura de grupos ao nível de sistema. Os problemas típicos de integração prendem-se com: a existência de *browsers*, que não são totalmente compatíveis com as diferentes tecnologias e que apresentam diferentes comportamentos em diferentes plataformas; e com uma implementação incompleta das normas e protocolos.

No caso particular destas aplicações acrescenta-se ainda a necessidade de ter uma resposta por parte da interface que não esteja condicionada pelo tradicional processo de refrescamento da página Web. O utilizador deve ter uma interacção com o sistema que não o obrigue a, explicitamente, ter que desencadear acções relacionadas com a actualização da informação. Essa é no fundo a motivação para a utilização conjunta de HTML, Javascript e Ajax.

6.1.2 Sistema de informação

Nos domínios de aplicação considerados e já referidos no capítulo 2, existe de um modo geral a necessidade de definir um sistema de informação onde é mantida a informação produzida e trocada entre os diversos utilizadores do sistema. Embora a existência explícita de um sistema de informação não seja uma exigência de todas as aplicações, de um modo geral tal é necessário, como mecanismo para armazenar e estruturar os dados manipulados. A informação e o seu modelo de dados é obviamente dependente e parte integrante da aplicação, pelo que é da responsabilidade do programador, a sua definição e implementação.

Ao nível da arquitectura do modelo, como foi referido no capítulo 5, definiu-se e caracterizou-se um conjunto de primitivas que permitem a gestão e o acesso aos dados. Essas primitivas caracterizam uma interface genérica, sendo no entanto a sua construção dependente e da responsabilidade da infraestrutura de suporte de dados considerada. Ou seja, o suporte de dados deve disponibilizar as funcionalidades que permitam a manipulação dos dados armazenados[2]. O suporte de dados, ou seja o sistema de informação (SI) utilizado, pode tomar diferentes formas, desde uma base de dados relacional a um sistema de armazenamento de dados em ficheiros de organização simples, dependendo obviamente das necessidades da aplicação e das opções tomadas pelo programador da mesma.

No caso da aplicação protótipo, que irá ser descrita mais à frente na secção 6.4, a infraestrutura de suporte de dados utilizada foi uma base de dados relacional ORACLE [McL07], tendo as primitivas de acesso e manipulação dos dados sido implementadas em Java e Oracle PL/SQL.

6.2 Principais funcionalidades de suporte às aplicações

Da análise das necessidades das diversas aplicações consideradas (capítulo 2) foi identificado um conjunto de funcionalidades típicas que o modelo pretende comportar.

Essas funcionalidades foram caracterizadas, em diferentes áreas, de acordo com as necessidades mais habituais das aplicações.

- entrada e registo de utilizadores no sistema;
- gestão de perfil de utilizadores e de grupos;
- gestão de grupos (criação/remoção/constituição);
- comunicação no grupo;
- produção e consulta de dados partilhados dentro de grupos;

[2]De um modo geral e em grande parte das aplicações analisadas, são necessários mecanismos para inserir, consultar e remover dados de diferentes tipos.

- coordenação de tarefas dentro do grupo.

Estas funcionalidades e a forma como são integradas com o modelo proposto, são descritas de seguida.

6.2.1 Entrada e registo de utilizadores no sistema

De um modo geral, as aplicações consideradas (capítulo 2) assim como os cenários de utilização descritos na secção 4.7, necessitam de uma funcionalidade que lhes permita validar a entrada de elementos no sistema. Essa entrada pode ser mais ou menos complexa, de acordo com as características da aplicação e o nível de segurança de acesso pretendido. Numa versão simples de controlo de acesso, um utilizador necessita de um identificador (nome) e de uma validação de acesso (chave). No entanto, mais dados podem ser pedidos de modo a definir o perfil do utilizador (pelo menos da primeira vez). Os dados exigidos e a obrigatoriedade do seu preenchimento são opções tomadas ao nível da aplicação, sendo a sua gestão efectuada pelo sistema de dados próprio da aplicação. Para a entrada e registo de um utilizador, sendo este modelado como uma entidade elementar[3], é necessário proceder ao seguinte conjunto de acções.

De seguida é apresentada, em pseudo código, a sequência de procedimentos que devem ser executados para efectuar o registo de um novo elemento.

```
...
me = new MagoEntity(...);                // cria objecto mago
usi = new AppSIType(...);                // cria objecto para interacção com o SI
...
{ pedido de nome (login) e consulta SI}
{ se (login) válido}
        { cria entrada no SI }
        pwd = { pede ao utilizador dados definidos pela aplicação, caso mais simples
            pede chave }
        attribs = pwd;
        id = me.register(login, attribs; out: identif)
        if ( id !=  null )
                // registo efectuado com sucesso

                // cria entrada da nova entidade
                usi.si_newEntity(login, pwd; out: res);

                if ( res )
                        // criação de entrada no SI efectuada com sucesso

                        // actualiza info no SI dados referentes ao novo utilizador
                        user_info = { lê informação referente ao novo membro }
                        usi.si_setEntity( login, user_info; out: res2 );

                        if ( res2 )
                                // actualização de dados do user no SI efectuada
                                // com sucesso
```

[3]Por normalmente um utilizador ser modelado como uma entidade elementar, ao longo deste capítulo esses dois termos possuem um mesmo significado.

```
                        // processo de configuração de grupos implícitos
                        updateProfile( login, userInfo; out: res2 );
                    else
                        // processa erro de actualização de dados de
                            utilizador
                else
                    // erro, não foi possível criar entrada no SI
                    // remove entidade do sistema
                    me.unregister(id);
            else
                // erro, não foi possível efectuar o registo do utilizador
                ...
```

Listagem 6.1: Registo de um novo utilizador

Após a execução deste troço de código, o elemento torna-se membro do sistema, passando a pertencer ao "Grupo Universal". Como já foi referido anteriormente, o grupo Universal é um grupo pré-definido, ao qual todos os utilizadores pertencem, que é gerido pelo servidor GU que suporta todas as interacções de entrada e saída de utilizadores assim como a criação e remoção de grupos (secção 6.3.2). O "lançamento" deste servidor é efectuado quando da configuração/arranque das aplicações.

6.2.2 Gestão de perfil de utilizadores e de grupos

Este tipo de funcionalidades permite efectuar a gestão de utilizadores, definindo o conjunto de atributos e configurações que permitem caracterizar e identificar, de forma inequívoca, cada elemento. A partir do conjunto de atributos configurado e instanciado de cada utilizador, é possível desencadear processos de formação de novas formas de agregação e estruturação de elementos, podendo formar "comunidades" de utilizadores que possuem um mesmo subconjunto de propriedades/atributos. Em diversas aplicações, a definição de perfis de utilizador torna possível a personalização da forma de interacção entre o utilizador e a aplicação, por exemplo, ao nível da forma de visualização de informação. A manipulação deste tipo de informação, por parte das aplicações, não está directamente relacionada com o modelo de suporte de grupos mas sim com o sistema de informação (SI) responsável pela gestão dos dados da aplicação e com o seu modelo de dados. As aplicações manipulam os seus dados próprios através de procedimentos para o acesso directo ao SI. No capítulo 5, foram definidos alguns métodos genéricos, que podem ser adaptados pelas aplicações, que permitem manipular a informação referente ao elementos presentes no sistema. Como exemplo, para a actualização dos dados (atributos e propriedades) de uma entidade elementar (utilizador) no sistema de informação, pode ser executada a seguinte sequência de acções:

```
...
// Tipo definido pela aplicação depende da estrutura definida para a nova entidade
userInfo = { lê informação referente aos atributos e propriedades do novo membro }
```

```
usi.si_setEntity( login, userInfo; out: res );

if ( res )
        // actualização de dados no SI efectuada com sucesso

        // procede a reconfiguração dos grupos implícitos
        // (no servidor Grupos Implícitos)
        updateProfile(login, userInfo; out: res2);

        if ( res2 )
                // constituição dos grupos implícitos actualizada
        else
                // processa erro de actualização de grupos implícitos
else
        // processa erro de actualização de dados da entidade
...
```

Listagem 6.2: Exemplo de actualização do perfil de utilizador

É da responsabilidade da aplicação definir os dados, assim como a sua política de preenchimento e validação. A actualização dos dados desencadeia, ao nível do sistema, uma reconfiguração da constituição dos grupos implícitos que é efectuada pela função $updateProfile()$.

6.2.3 Gestão de grupos

A este nível, as aplicações têm necessidades bastante diversificadas, dependendo da filosofia e ambiente de utilização, bem como do tipo de utilizadores a que se destinam. De um modo geral, para a gestão de grupos, pretende-se que sejam disponibilizadas funcionalidades que permitam efectuar a criação e destruição de grupos e gerir as formas de acesso ao grupos existentes. As primitivas disponibilizadas pelo modelo são flexíveis relativamente às diferentes estratégias de formação dos grupos e podem facilmente adaptadas ao tipo de utilização pretendida. Consideram-se três áreas distintas de funcionalidades pretendidas: (1) criação; (2) destruição; (3) filiação.

(1) Criação: O processo de criação de um grupo no sistema e a sua actualização ao nível do sistema de informação, obedece à seguinte sequência de acções:

```
...

// cria entrada para o novo grupo no SI
// login identificador da entidade/utilizador criadora do grupo
si.newGroup( nome_grupo, login; out: res );

if ( res )
        // criação do grupo no SI efectuada com sucesso

        ...
        // { PROCESSO DE CRIAÇÃO DE GRUPO NO MAGO }

        // por invocação da primitiva create que retorna o valor valida com indicação
```

```
        // do sucesso (ou não) da operação
        ...

        if ( valida )
                // criação de grupo efectuada com sucesso

                lista_param = {parametros de configuração do grupo};

                // procede a actualização de dados do grupo no SI
                si.si_setGroup( nome_grupo, id_grupo, lista_param );

        else
                // processa erro de criação do grupo no sistema
                // pode desencadear processo de remoção da entrada do grupo
                // no SI através de si_removeGroup( id_grupo, resultado );
else
        // processa erro de criação de novo grupo no SI
...
```

Listagem 6.3: Exemplo de criação de um grupo por parte de um utilizador

Os diferentes tipos de aplicações e modos de utilização existentes, levam a que a formação de novos grupos possa obedecer a diferentes critérios, de acordo com a sua especificidade, pelo que a invocação da primitiva *create*() pode ter diversas configurações. Por exemplo, podem ser formados grupos:

- com um número limite de membros

```
...
// { PROCESSO DE CRIAÇÃO DE GRUPO NO MAGO }

max = { número máximo de membros do grupo };
valida = me.create( id_criador, nome_grupo, EXPLICIT, - , max, -; out: id_grupo );
...
```

- tendo por base um conjunto pré-definido de membros, ou seja com uma lista de participantes explícita

```
...
// { PROCESSO DE CRIAÇÃO DE GRUPO NO MAGO }

// criação lista de participantes com a estrutura [identif,chave]
lista = { define lista de membros };
max = sizeOf( lista );
valida = me.create( id_criador, nome_grupo, EXPLICIT, BY_LIST, max, lista; out:
    id_grupo );
...
```

- sem restrições, ou seja, a entrada de novos membros não é sujeita a nenhum processo de avaliação

```
...
// { PROCESSO DE CRIAÇÃO DE GRUPO NO MAGO }

valida = me.create( id_criador, nome_grupo, EXPLICIT, -, -, -; out: id_grupo );
...
```

- sujeito a um processo de votação, que segue uma política de maioria:

```
...
// { PROCESSO DE CRIAÇÃO DE GRUPO NO MAGO }

valid = me.create( id_creator, nome_grupo, EXPLICIT, MAJORITY, -, -: out: id_grupo );
...
```

- sujeito a um processo de votação, em que basta um dos membros votar favoravelmente:

```
...
// { PROCESSO DE CRIAÇÃO DE GRUPO NO MAGO }

valid = me.create( id_creator, nome_grupo, EXPLICIT, BY_ONE, -, -; out: id_grupo );
...
```

- baseados na(s) característica(s) que são verificadas pelos participantes (grupos implícitos)

```
...
// { PROCESSO DE CRIAÇÃO DE GRUPO NO MAGO }

lista_params = { critérios de formação do grupo }
valida = me.create( id_criador, nome_grupo, IMPLICIT, - , -, lista_params; out:
    id_grupo );
...
```

(2) Destruição: Ao nível da destruição de grupos, os padrões de utilização existentes em diversas aplicações prendem-se com a forma como é efectuada a notificação e saída dos elementos filiados no grupo. Diferentes formas de destruição de grupos podem ser consideradas também de acordo com a configuração inicial do grupo (criação). Ao remover um grupo diversas políticas podem ser seguidas:

- destruir grupo sem aviso prévio aos membros

```
...
valida = me.destroy( id_grupo, nome_grupo, NO_NOTIFICATION );
...
```

- destruir grupo após emitir aviso de remoção para os membros

```
...
valida = me.destroy( id_grupo, nome_grupo, NOTIFICATION );
...
```

- emitir aviso e aguardar confirmação, por parte de todos os membros, da recepção do aviso antes de proceder à destruição

```
...
valida = me.destroy( id_grupo, nome_grupo, NOTIFICATION_ACK );
...
```

- destruir grupo, somente se este se encontrar vazio, ou seja sem membros

```
...
if (number_member == 0 )
      valida = me.destroy( id_grupo, nome_grupo, NO_NOTIFICATION );
...
```

Após a destruição do grupo pode-se proceder à remoção do grupo no SI através da chamada de *si_removeGroup*(*id_grupo*;*out* : *res*). Esta trata-se de uma opção da aplicação que pode optar por manter a informação referente ao grupo no SI, que pode passar a funcionar como um histórico de grupos já existentes.

(3) Filiação: A filiação em grupos, é de um modo geral, uma acção explícita por parte do utilizador, podendo este processo ser ou não validado pelo grupo, ou seja, a entrada de um novo membro pode ou não ser dependente da aceitação por parte do grupo enquanto entidade colectiva. Para se tornar membro de um grupo, o utilizador desencadeia um processo de filiação ao invocar a primitiva de *join*.

```
...
info_msg = { informação de acesso fornecida pela entidade }
valida = me.join( my_id, grupo_id, info_msg; out: entrada_grupo_id );
// entrada_grupo_id identificação da entidade no novo grupo a que se filiou

if ( valida )
        // actualiza no SI a filiação do membro no grupo
        si.putGroupMember( grupo_id, my_id );
        if ( res )
                // processo de filiação concluído
        else
                // processa erro de actualização de filiação no SI pode optar por:
                //  - repetir tentativa de actualização de filiação no SI
                //  - abortar o processo de filiação removendo entidade do grupo me.
                    leave(my_id, grupo_id)
else
        // processa erro de filiação
...
```

A política de validação do pedido de acesso *join*(), depende do critério que foi definido quando da criação do grupo, sendo esta uma característica estática da configuração do grupo.

6.2.4 Comunicação no grupo

A comunicação desempenha um papel fundamental na estruturação e funcionamento de estruturas de grupo e para a formação de comunidades de utilizadores que partilham um mesmo tipo de interesses. Existem diversas formas possíveis para o estabelecimento de interacções dependendo estas, em grande parte, da filosofia e tipo de utilização pretendidos. Ao existir a possibilidade de definir estruturas de grupos de utilizadores, passamos a ter formas de comunicação não só entre utilizadores mas

também com a "entidade" colectiva de utilizadores (grupos). Assim a um nível de estruturação superior, podemos ter, como pode ser observado na figura 6.2, comunicação directa entre os diversos intervenientes.

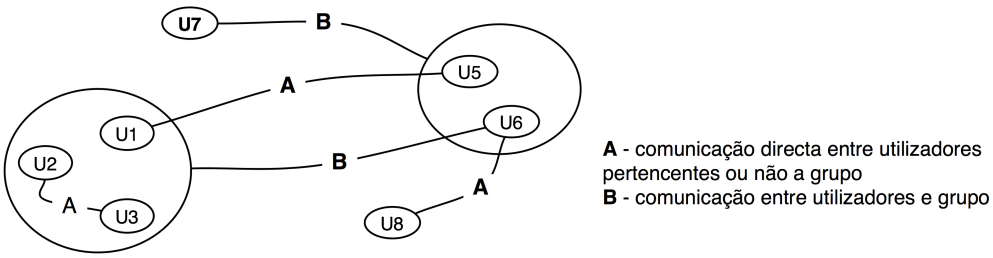

Figura 6.2: Interacções directas possíveis entre os diversos intervenientes

Na implementação das interacções, podem ser utilizados diferentes modelos de comunicação, tais como, troca de mensagens, eventos ou espaço partilhado. A natureza das aplicações pode determinar a forma de comunicação entre os diversos intervenientes. Na aplicações analisadas, verificou-se que o estabelecimento de interacções entre os utilizadores possuem, tipicamente, um carácter assíncrono.

Os mecanismos de comunicação existentes no modelo proposto, permitem estabelecer distintos tipos de interacções, que são de seguida ilustrados.

1) Comunicação directa entre elementos, onde é explicitamente nomeado o interlocutor:

- envio de mensagem para um utilizador, seguida de mensagem dirigida a um grupo

```
...
valid = me.send( my_id, receiver_id, "olá, envio de msg1 para utilizador" );
...
valid =  me.send( my_id, grupo_id, NOTIFY, "olá, envio de msg2 para grupo" );
...
```

- leitura de uma mensagem, por parte de utilizador, de forma não bloqueante

```
...
valid = me.receive( NO_BLOQ, msg1 );
...
```

- leitura, por parte do utilizador, de uma mensagem dirigida a um dos grupos de que é membro

```
...
// membros do grupo ao qual foi dirigida a mensagem receberam um aviso
// de chegada da mesma
// a mensagem é identificada e colocada na fila global do grupo

// este processo de receive desencadeia processo de leitura dessa mensagem
valid = me.receive( NO_BLOQ, id_msg, grupo_id, msg );
...
```

2) Comunicação através do mecanismo de eventos:

Este é o mecanismo utilizado quando se pretende a difusão de informação dentro de um grupo.

Existem duas fases do lado do produtor de eventos: uma, em que se "cria" e anuncia (*advertise()*) o evento ao grupo e outra, em que se desencadeia o evento (difunde). Também do lado do consumidor de eventos existe uma fase prévia, em que este deve primeiro subscrever o evento e somente depois fica "sensível" à ocorrência desse tipo de evento. Do ponto de vista de uma aplicação, se pretender definir um novo evento deverá criar o novo tipo de evento, sendo este caracterizado por um identificador, por um conjunto de atributos e por um método de tratamento que é definida pelo anunciante (produtor) do evento. A subscrição por parte dos consumidores corresponde a obter a informação que lhe permite identificar e desencadear o tratamento definido para o evento específico. Existem no entanto eventos, ditos de sistema, cujos métodos de tratamento se encontram já pre-definidos na implementação do modelo, conforme foi referido no capítulo 5.

De seguida é apresentada a sequência de acções a serem desencadeadas por parte do produtor e dos consumidores:

- produtor

```
...
// associa um identificador (tipo) ao evento
ev_type = { cria identificador de evento };

// anúncio do tipo de evento por parte de um produtor
adv_type = PASSIVE;  // tipo de anúncio de novo tipo de evento

me.advertise( my_id, group_id, PASSIVE, ev_type, info );
...
// ocorrência do evento é desencadeada pelo produtor
// definição do evento
...

// criação de um evento a ser publicado no grupo
evento  = new MGEventType(...);
me.publish( group_id, evento );
...
```

- consumidor

```
...
// consulta de eventos anunciados no grupo
me.check_adv( group_id, event_list );
// em event_list são indicados os eventos definidos até ao momento
// cada elemento da lista possui a seguinte informação [tipo_evento, id_produtor]
...
// pode optar por subscrever um determinado tipo de evento de determinado grupo
// especificando o identificador do evento e o seu método de
// atendimento, executado quando da ocorrência desse evento
me.subscribe( group_id, tipo_evento, metodo_atendimento );
...
```

3) Comunicação através do espaço partilhado:

Este é o mecanismo utilizado quando se pretende a partilha de informação, em particular conteúdos, entre os membros de um grupo.

O modelo proposto disponibiliza um conjunto de primitivas que permitem colocar dados (*update*), aceder (*consult* e *get*) e procurar todos os dados que obedeçam a um determinado padrão (*find*). Neste tipo de comunicação existe uma estreita dependência da forma como os dados são armazenados no sistema de informação.

É apresentada de seguida a sequência de acções a desencadear por quem produz os conteúdos para o grupo e por quem pretende consultar e aceder aos mesmos de forma simples e directa:

- produtor de conteúdo

```
...
// estes dados pode ser por exemplo
obj_conteudo = { construção do objecto a ser armazenado no SI };
si.si_updateCGroup( id_grupo, my_id, obj_conteudo; out: data_id );

// constrói dados a serem colocados no espaço partilhado do grupo,
// por exemplo, conjunto de tags que caracterizem o conteúdo
// e forma de acesso aos mesmos no si (localização)
data = new TupleDataType(dados + data_id);
data_acess = PUBLIC;
me.update( my_id, id_grupo, PUBLIC, NO_NOTIFY, data );
...
```

- acesso (leitura não destrutiva) a dados do espaço com base em valores de pesquisa definidos pela aplicação

```
...
// constrói objecto com informação referente aos dados que pretende pesquisar no
     espaço
// sendo este dependente da aplicação
dados_pesquisa = new AppType(...);
...
// constrói tuplo de pesquisa
dataP = new TupleDataType(dados_pesquisa);

// leitura não destrutiva de dados (tuplo) do espaço partilhado de um grupo
data_lida = new TupleDataType();
me.consult( my_id, id_grupo, produtor_id, -, NO_BLOCK, dataP; out: data_lida );

// obtem referência aos dados a partir da data_lida
data_id = data_lida.getRef();
// no caso da informação se encontrar no SI
// vai buscar o conteúdo com identificação data_id ao SI
si.si_getCGroup( id_grupo, data_id; out: objecto_dados );
...
```

- acesso a dados do espaço, a partir da primitiva de pesquisa $find()$

```
...
// constrói objecto com informação referente aos dados que pretende pesquisar no
     espaço
```

```
dados_pesquisa = new AppType(...);
...
// constrói tuplo de pesquisa
dataP = new TupleDataType(dados_pesquisa);

// leitura não destrutiva de dados (tuplo) do espaço partilhado de um grupo
// para estrutura local
me.find( my_id, id_grupo, produtor_id, -, NO_BLOCK, dataP; out: num_dados );

// leitura de todos os tuplos da estrutura local que foi preenchida pela
    primitiva $find$
data_lida = new TupleDataType();
for i=1 to num_dados
        me.consult( i; out: data_lida );

        // acesso ao conteúdo no SI

        // obtem referência aos dados a partir da data_lida
        data_id = data_lida.getRef();
        si.se_getCGroup( id_grupo, data_id; out: objecto_dados);
        ...
...
```

- remoção de dados com base em valores de pesquisa definidos pela aplicação

```
...
// constrói objecto com informação especificada pela aplicação
// referente aos dados que pretende pesquisar no espaço
dados_pesquisa = new AppType(...);
...
// constrói tuplo de pesquisa
dataP = new TupleDataType(dados_pesquisa);

// leitura destrutiva de dados (tuplo) do espaço partilhado de um grupo
data_lida = new TupleDataType();
me.get( my_id, id_grupo, produtor_id, -, NO_BLOCK, dataP; out: data_lida );
...

// remoção dos dados do SI
// obtem referência aos dados a partir da data_lida
data_id = data_lida.getRef();

// no caso da informação se encontrar no SI
// vai buscar o conteúdo com identificação data_id ao SI
si.si_getCGroup( id_grupo, data_id; out: objecto_dados );

// remove o conteúdo com identificação data_id ao SI
si.si_removeCGroup( id_grupo, data_id; out: res );
...
```

6.2.5 Produção e consulta de dados partilhados

Um dos principais motivos que justifica a formação de estruturas de grupos é a possibilidade de partilha de dados dentro dessa organização. Esses dados podem ser dos mais diversos tipos e a forma de acesso permitida aos utilizadores é dependente das permissões definidas e configuradas pela aplicação. De um modo geral, a partir do

momento em que um utilizador se torna membro de uma estrutura de grupo, passa a ter acesso aos dados e serviços disponíveis que foram definidos para esse grupo, podendo ou não tornar-se também ele produtor de dados.

Um grande número de aplicações necessita de definir estruturas, com conjunto de permissões e políticas de acesso bem definidas, para acesso a dados provenientes de diferentes utilizadores. De um modo geral, o conjunto de funcionalidades, ao nível da colocação e consulta de dados para partilha por um determinado conjunto de utilizadores, é bastante semelhante entre si, sendo as diferenças colocadas mais ao nível da filosofia subjacente à aplicação.

Por exemplo, em aplicações do tipo *blog* ou *forum*, o conjunto de materiais (textos, fotos, vídeos) disponibilizado é gerido por uma entidade com privilégios de administração, que tem a seu cargo a validação dos dados que são enviados para partilha pelo conjunto de participantes. De um modo geral, neste tipo de aplicações, as permissões de acesso ao conjunto de dados são limitadas, em particular, no que se refere à possibilidade de modificação e de colocação de nova informação. Já o acesso para consulta é bastante mais "aberto", em particular em aplicações Web onde o acesso à informação (pelo menos de parte dela), é disponibilizado a toda a comunidade.

Existe no entanto outro conjunto de aplicações onde a informação e o conjunto de dados disponíveis vão sendo construídos, de forma incremental pelos diversos participantes, sendo todos eles responsáveis pelos dados partilhados. Um dos exemplos deste tipo de aplicações é o Geni [Gen07] no qual é construída uma estrutura de grupos familiares cujos membros são convidados a participar, através da colocação (e/ou alteração) de dados e conteúdos, aumentando desta forma o conjunto de informação disponibilizada a toda a estrutura familiar. Outros exemplos deste tipo podem ser encontrados como funcionalidades disponibilizadas em diferentes comunidades. Por exemplo, ao definir grupos de interesse dentro de algumas das comunidades mais genéricas passa-se a ter um grupo mais fechado de participantes, onde existe um espaço mais restrito ao qual, após validar a sua entrada, os membros podem ter acesso e disponibilizarem informação de uma forma mais controlada.

Tendo por base algumas das primitivas do modelo, foi já exemplificada, na subsecção anterior, a utilização por parte de uma aplicação, do espaço partilhado como mecanismo de partilha de conteúdos dentro de um grupo. Para a partilha de dados, recorrendo ao espaço partilhado oferecido pelo modelo, podem ser definidas diversas políticas de divulgação e de acesso aos conteúdos partilhado, sendo a sua escolha uma opção da aplicação.

6.2.6 Coordenação de tarefas

A formação de estruturas de grupo tem também, muitas vezes, como objectivo o desempenhar de determinada tarefa de uma forma colectiva, o que leva a necessidades

de coordenação entre os diversos participantes. Existem diversas situações em que as aplicações podem necessitar de desencadear processos de coordenação e de sincronização, entre as acções que estão a ser executadas pelos seus utilizadores.

Isto pode acontecer em aplicações com situações como as que surgem em jogos de múltiplos utilizadores ou, com um exemplo mais concreto, em leilões *online*. Em jogo, pode ocorrer um grande número de situações, onde existe a necessidade de coordenação entre as acções que estão a ser desencadeadas pelos vários participantes, como por exemplo: necessidade de um número definido de jogadores se registar, antes de dar início ao jogo; aguardar que todos os jogadores da equipa ultrapassem determinado obstáculo, antes de fazer evoluir o jogo; recolher objectos existentes distribuídos pelo espaço, antes das restantes equipas. No caso dos leilões *online*, existe a necessidade de coordenar os lances que vão sendo efectuados pelos diversos participantes, de modo a garantir que todos têm a possibilidades de participar com base em informação actualizada dos lances já realizados.

O modelo proposto apresenta diversos mecanismos que permitem ao programador da aplicação implementar a coordenação dentro de um grupo. Pode-se optar por: utilização de mensagens directas, para que diversos utilizadores coordenem as suas actividades; difusão de mensagem pelos membros de grupo; a utilização do espaço partilhado; ou uma combinação dos diversos mecanismos de interacção disponibilizados.

A figura 6.3 resume o conjunto de funcionalidades referidas e que, tipicamente, cobrem o conjunto de necessidades das aplicações analisadas. A figura ilustra também a relação dessas funcionalidades, não só com o modelo MAGO, mas também com o sistema de informação.

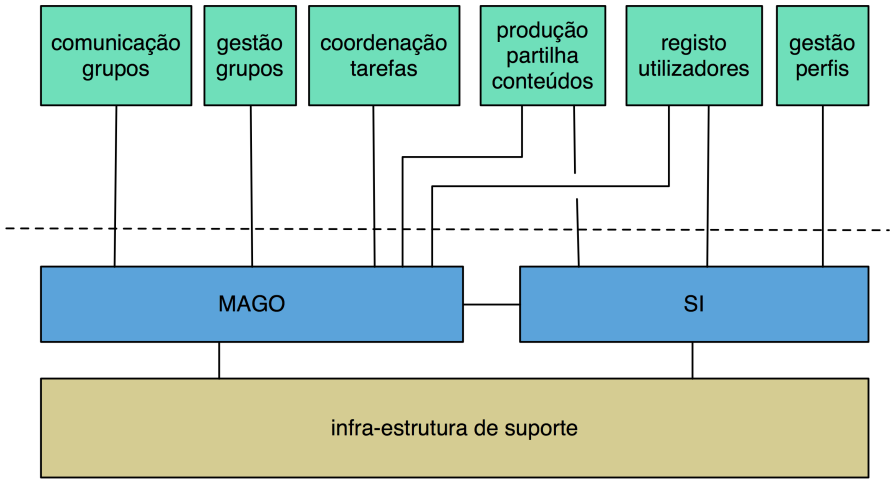

Figura 6.3: Classes de funcionalidades e a sua relação com o sistema de suporte

6.3 Organização das aplicações baseadas no modelo

A organização das aplicações baseadas no modelo MAGO assenta em interacções do tipo cliente-servidor que dão acesso a um conjunto de serviços que realizam as funcionalidades definidas no modelo.

As interacções entre utilizadores são geridas ao nível de um servidor (ou conjunto de servidores) responsável por mediar as comunicações entre utilizadores.

Outra estrutura de comunicação passa pela possibilidade de comunicação directa entre os utilizadores ou seja ponto a ponto, permitindo assim descentralizar o processo de comunicação o que se reflecte também num aumento de robustez em casos de falha.

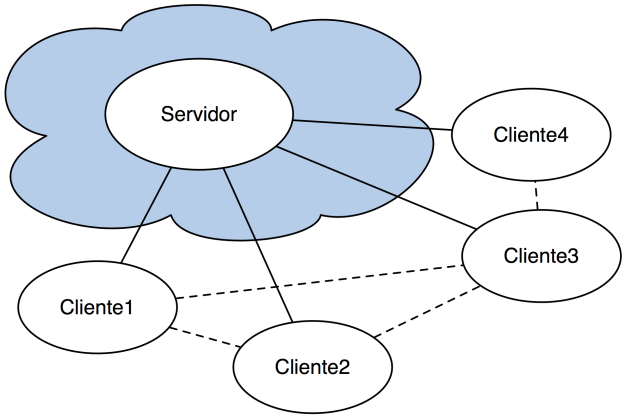

Figura 6.4: Organização cliente-servidor e ponto a ponto

Na situação ilustrada na figura 6.4, o servidor desencadeia o processo de estabelecimento de comunicação entre os clientes, diponibilizando-lhes informação que lhes permite (posteriormente) comunicarem entre si, conjugando, desta forma, os dois modelos de comunicação, cliente-servidor e directa (ponto a ponto).

Ao nível da arquitectura apresentada, todos os clientes (utilizadores) se encontram numa mesma estrutura que foi denominada de grupo Universal, sendo a gestão de comunicações efectuada dentro deste espaço, o que facilita a forma como diferentes clientes podem ser reconhecidos e identificados entre si.

O facto de as aplicações terem, em alguns casos, de lidar com dispositivos heterogéneos, possivelmente com pouca capacidade computacional, condiciona fortemente a forma como é desenvolvida a arquitectura das mesmas. A necessidade de integrar um número, o mais alargado possível de dispositivos, conduz a uma organização do tipo apresentado na figura 6.5. Neste tipo de arquitectura, o "cliente" é composto por dois elementos distintos, um responsável pelas comunicações e comportamento geral da aplicação e outro encarregue simplesmente da interface com o utilizador e da gestão dos elementos visuais presentes na interface.

Ao efectuar esta separação, fica em aberto a possibilidade dos dois elementos coabitarem num mesmo suporte computacional ou de cada um deles se encontrar fisica-

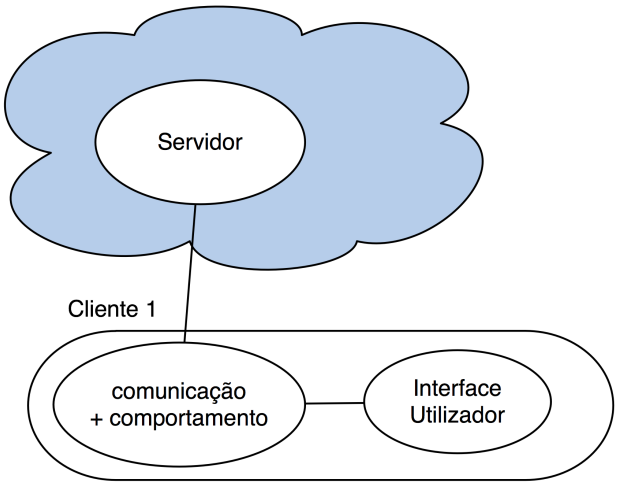

Figura 6.5: Cliente decomposto nos módulos de execução e de interface com o utilizador

mente separado, tornando possível que um dispositivo móvel de pequena dimensão (telemóvel ou equivalente) possa assumir o papel do dispositivo gestor da interface com o utilizador.

Esta separação obriga a que a maior ênfase seja dada ao protocolo de comunicação entre os dois módulos, algo que não era particularmente relevante até então. Os dados necessários à gestão da interface de utilizador deixam de estar "presentes" no contexto da aplicação/dispositivo, que vai necessitar de comunicar com o elemento responsável pelas comunicações, para actualizar o estado daquela interface e para fazer reflectir, no sistema, a interacção do utilizador com a interface cliente.

Uma forma de alargar o mais possível o espectro de dispositivos, passíveis de serem utilizados como interface de utilizador, foi a de adoptar o protocolo HTTP, como base da comunicação entre os módulos clientes, podendo assim ser suportados quaisquer dispositivos capazes de executar um Web browser[4].

A organização (figura 6.6) pode passar a ser vista como possuindo três tipos de objectos, em que o servidor e os elementos de comunicação do cliente (denotados por *Ci* na figura) formam um núcleo, a que se liga um conjunto de aplicações de interfaces do lado do cliente (denotados por *Ci* http).

Do ponto de vista de um observador externo, o sistema comporta-se como reagindo unicamente a pedidos HTTP, já que na orla exterior do sistema, se encontram apenas aplicações Web, comunicando com o sistema, de uma forma em tudo idêntica à forma como efectuam os seus acessos a sites Web.

[4]Pode ser um Web browser ou aplicação equivalente por exemplo uma aplicação do tipo *Flash* que pode substituir o browser, sempre que a reduzida capacidade de *scripting* ao nível do browser for limitativa para a implementação das funcionalidades de que a aplicação necessita.

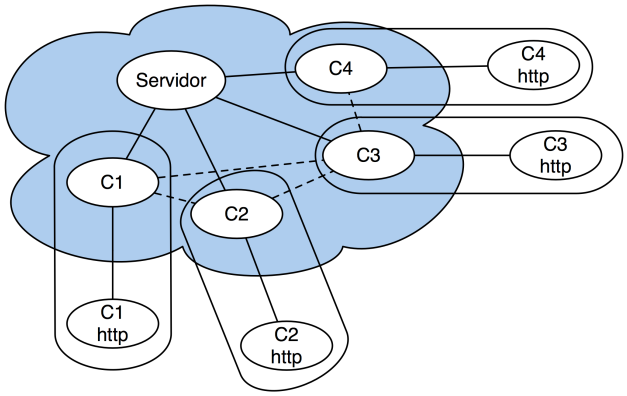

Figura 6.6: Organização com clientes decompostos

6.4 Integração do modelo numa aplicação protótipo

O protótipo que aqui se descreve foi desenvolvido com o objectivo de testar e validar o modelo proposto, em particular o modo de utilização das funcionalidades disponibilizadas por este. Foi para esse efeito, desenvolvida uma aplicação genérica, que disponibiliza aos utilizadores uma interface cliente, através da qual estes podem comunicar no contexto de grupos (serviço do tipo *chat*), acedendo e disponibilizando dados que podem ser partilhados dentro de cada um dos grupos a que pertencem. Os utilizadores podem ainda configurar os seus dados pessoais, criar novos grupos e juntar-se a grupos já existentes. De seguida, é apresentada uma descrição mais detalhada das funcionalidades desenvolvidas, assim como a sua implementação com base na arquitectura do modelo apresentada no capítulo 5.

6.4.1 Descrição da interface do cliente

No desenvolvimento desta aplicação, foram considerados como principais objectivos, como já foi referido, a validação do modelo proposto e a motivação para a criação e desenvolvimento de novas funcionalidades baseadas em grupos de utilizadores. Para além disso, esta aplicação constituiu também uma plataforma de estudo da forma como proceder à integração dos mecanismo do modelo, com as restantes tecnologias necessárias à implementação das aplicações.

Toda a concepção do protótipo e das funcionalidades disponibilizadas por este, foram assim definidas de modo a tirar o máximo partido da estruturação e padrões de utilização oferecidos pelo modelo de grupos. Algumas das funcionalidades desenvolvidas para este protótipo foram elaboradas de modo a poderem vir a ser incluídas como serviços noutras aplicações já existentes, como por exemplo, o *chat* de grupo ou o contexto de grupo (partilha de conteúdos por parte de membros de um grupo).

Ao longo do desenvolvimento do protótipo partiu-se do pressuposto de que este

iria ser utilizado num ambiente onde cada utilizador possui um dispositivo[5] através do qual acede a uma interface, que lhe permite gerir e visualizar os grupos a que pertence e os respectivos conteúdos. Cada utilizador (cliente) pode estar envolvido de forma explícita em diversos grupos, podendo também agregar-se indirectamente em grupos de interesse (grupos implícitos). Estes últimos grupos, no caso do protótipo, são geridos por um utilizador especial, o administrador, que tem acesso a uma aplicação/interface específica onde podem ser definidas e especificadas quais as características que estão na base da formação dos grupos implícitos (*System Groups*).

Os utilizadores/clientes têm acesso a uma interface simples, em que sobressaem três grandes grupos de funcionalidades:

- *My Groups* (grupos explícitos)
- *System Groups* (grupos implícitos)
- *User Profile* (perfis de utilizador e suas preferências)

Estes grupos de funcionalidades correspondem aos três ecrãs que estão disponíveis na forma de *tabbed folders* (figura 6.7).

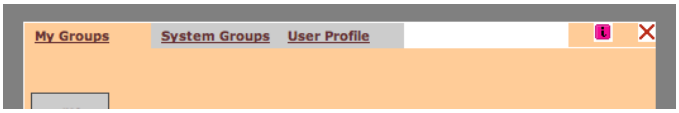

Figura 6.7: Grupos gerais de funcionalidades da aplicação disponibilizada aos clientes

Antes de aceder a esta zona da interface, o utilizador passou por um processo de autenticação simples, em que lhe foi pedido um nome (*login*) e uma chave para acesso ao sistema. Ao chegar aqui, o utilizador passa a poder visualizar e ter acesso aos seus grupos (a que explicitamente aderiu) identificados na secção *My Groups*, aos grupos implícitos a que o sistema o associou e que estão identificados como *System Groups* e finalmente, à área em que pode manipular os seus dados pessoais e as suas preferências (*User Profile*).

Ao nível do modelo de grupos, cada "cliente"é definido como uma entidade elementar, estando por isso associado a um objecto da classe *MagoEntity*. O processo de autenticação, com o pedido de um *login* e uma palavra chave de acesso, traduz-se num registo no sistema (*register*()).

My Groups (grupos explícitos)

Nesta área, o utilizador tem disponível a lista dos seus grupos explícitos, ou seja, pode visualizar a lista de grupos a que explicitamente se associou. De notar que, se o utilizador ainda não tiver explicitamente aderido a (ou criado) nenhum grupo, a lista

[5]O dispositivo deverá ter a capacidade de executar um browser.

apresenta-se vazia. A partir do momento em que o utilizador se filia explicitamente num grupo (ou seja, desencadeia o *join*()), esse grupo passa a surgir, sempre que o utilizador entra no sistema, deixando somente de surgir em duas situações: o utilizador explicitamente optou por sair do grupo (*leave*()) ou o grupo foi destruído (*destroy*()).

Ao seleccionar um grupo (seleccionando o seu nome), ficam visíveis, no ecrã, os membros que o constituem (*membership*()), passando o utilizador a ter acesso ao conjunto de funcionalidades definido para os *My Groups*. A visualização dos membros dos grupos é actualizada quando ocorre uma alteração na constituição dos grupos.

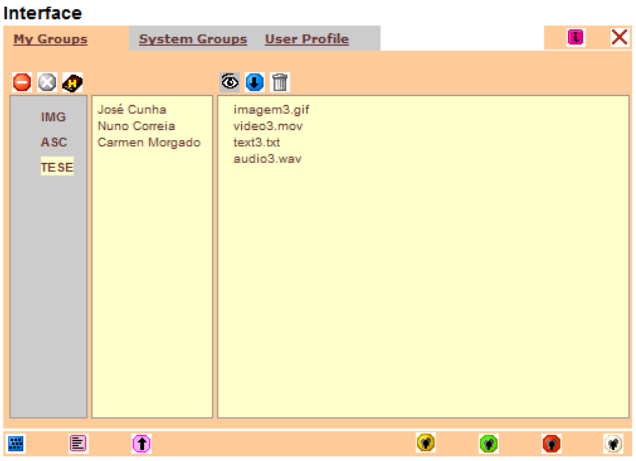

Figura 6.8: Menu com um grupo explícito seleccionado, com os seus membros e informação associada ao mesmo

Para além da constituição do grupo, é possível, através da interface, identificar quais os membros que se encontram actualmente *online* e *offline*, esta identificação feita por recurso a diferentes cores de acordo com o seu estado. A alteração do estado desencadeia uma chamada da primitiva *set_mode*(), por parte da aplicação.

Na zona inferior esquerda do menu, encontram-se os *icons* que permitem aceder às funcionalidades disponibilizadas para o grupo seleccionado, que se encontram descritos na tabela 6.1.

	contexto de grupo, dados pertencentes ao grupo (*context*)
	conversação de grupo (*chat*)
	carregamento de dados (*upload*)

Tabela 6.1: Funcionalidades disponíveis para os grupos a que pertence (procedeu à filiação)

Estas funcionalidades são aplicáveis sobre o grupo que foi antes explicitamente seleccionado pelo utilizador.

A selecção da opção "contexto" do grupo torna visível no ecrã (lado direito). A identificação dos documentos que os vários membros do grupo aí colocaram a constru-

ção desta lista é efectuada recorrendo às primitivas do modelo de pesquisa e consulta de dados partilhados do grupo (*find*()/*consult*()). Ao seleccionar um determinado ficheiro de dados, o utilizador pode optar por:

- visualizar o conteúdo do ficheiro (*consult*() e acesso ao SI);
- criar um cópia local do ficheiro (*consult*() e acesso ao SI);
- remover o ficheiro do espaço do grupo (*get*() e actualização da informação no SI).

A execução de cada uma destas opções é dependente das permissões que foram configuradas para o grupo.

A disponibilização de materiais para um grupo, ou seja, dos dados que passam a estar disponíveis no contexto do grupo, é efectuada através da selecção da opção de "carregamento" (*upload*) de ficheiros. A selecção da opção de *upload* activa uma interface tradicional de gestão de ficheiros que permite ao utilizador indicar o ficheiro que pretende colocar no contexto de grupo, desencadeando a actualização dos dados uma interacção com o sistema de informação através da chamada de métodos da sua interface (*si_updateCGroup*()) seguida da invocação da primitiva do modelo *update*().

Na opção de *chat*, é assumida a existência somente de *chat de grupo*, tendo esta opção sido tomada, devido a factores que se prendem com a usabilidade da aplicação, pelo facto de simplificar o desenho da interface disponibilizada e como a forma como esta é manipulada pelos utilizadores. Ou seja, a funcionalidade de *chat*, disponível neste protótipo, suporta exclusivamente uma conversação de grupo, sendo as mensagens acessíveis a todos os membros do grupo e no qual estes podem participar activamente. O mecanismo de comunicação do modelo aqui utilizado foi essencialmente o de comunicação directa com difusão, ou seja *send*() de mensagens com opção de SPREAD. Nesta situação o destinatário é o grupo (representante de grupo), que promove, através do mecanismo interno de eventos, a difusão das mensagens para os restantes membros do grupo. Os grupos podem ter acesso a um histórico[6] de comunicações mantidas. A opção de histórico é aplicada sobre o grupo seleccionado e permite visualizar as comunicações mantidas dentro do grupo até ao momento. Esta funcionalidade foi desenvolvida tendo por base os dados, referentes às mensagens, que são mantidos ao nível do sistema de informação da aplicação.

De salientar que, embora o modelo proposto suporte a comunicação directa entre membros de um grupo, essas funcionalidades não foram exploradas neste protótipo, tendo sido dado um especial ênfase às interacções em grupo e não entre indivíduos. Também no caso de partilha de materiais se encara o problema sempre do ponto de vista de partilha de conteúdos dentro de uma estrutura de grupo e não directamente

[6] A informação do "histórico" é mantida ao nível do sistema de informação pela aplicação.

entre entidades elementares.

No canto inferior direito, podem ser seleccionadas as funcionalidades, que se encontram descritas na tabela 6.2, referentes à gestão de grupos por parte dos clientes.

	configuração de grupo
	criação de grupos explícitos (My Groups)
	remoção de grupos explícitos
	visualização de grupos existentes para filiação

Tabela 6.2: Funcionalidades disponíveis para gestão de grupos

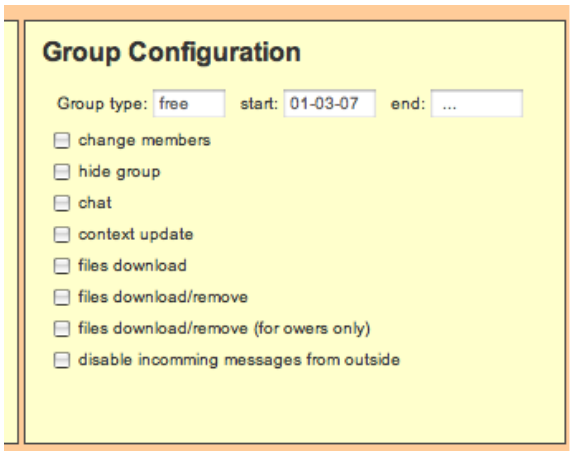

Figura 6.9: Menu de configuração de grupo disponibilizado pela aplicação

A opção de configuração do grupo permite visualizar e ajustar características de funcionamento do próprio grupo, de acordo com as permissões atribuídas ao utilizador e as funcionalidades que este tem disponíveis, ao nível da aplicação. Neste caso, temos como exemplo de configuração para o grupo seleccionado, o conjunto apresentado na figura 6.9.

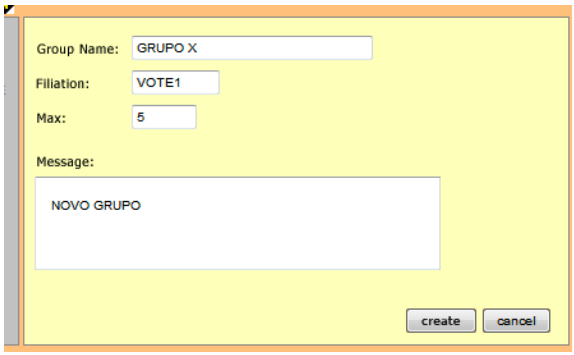

Figura 6.10: Menu para criação de grupos

Para a criação de um novo grupo (*create*()), o utilizador tem acesso a um menu (figura 6.10) através do qual pode especificar as características que pretende para o grupo. Através deste menu, o utilizador pode especificar elementos como o nome do grupo, a política de filiação de novos membros, as permissões e funcionalidades do grupo e o tempo de existência do grupo.

A destruição/remoção de um grupo permite ao utilizador através de um menu (figura 6.11), que este indique o tipo de remoção que pretende desencadear, podendo optar por emitir uma mensagem de notificação de destruição de grupo e proceder ao seu encerramento imediato (*destroy*() com opção de NOTIFICATION); ou por aguardar a confirmação de recepção da notificação por parte dos restantes membros activos do grupo (*destroy*() com opção de NOTIFICATION_ACK). Nestas duas formas de remoção de grupo é difundida informação para todos os membros activos do grupo através do mecanismo interno de eventos.

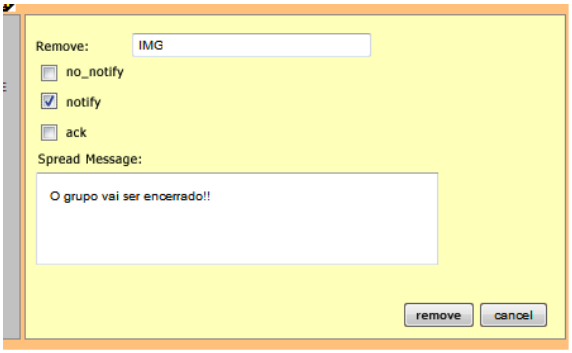

Figura 6.11: Menu para remoção de grupos

A opção de visualização de grupos apresenta, no ecrã, todos os grupos explícitos actualmente existentes no sistema, sendo esta informação obtida através do servidor de grupos. A partir desta interface de visualização, o utilizador pode seleccionar o grupo em que se pretende filiar (*join*()), sendo a validação da sua entrada efectuada com base na política de filiação que foi adoptada pelo grupo quando da sua criação.

System Groups (grupos implícitos)

A opção identificada como *System Groups* dá acesso aos grupos implícitos. A interface é idêntica à dos grupos explícitos e começa por apresentar a lista dos grupos implícitos de que o utilizador é membro.

A inclusão/filiação de membros nestes grupos é automática e baseia-se nas suas preferências, especificadas no seu perfil de utilizador. Neste tipo de grupos, o utilizador não tem controlo directo sobre os grupos a que pertence. Pode, no entanto, através das opções de *hidden* e *block*, controlar a forma como os restantes membros o vêem e interagem com ele.

	*block*	bloqueia as comunicações que lhe são dirigidas provenientes do grupo seleccionado
	*hidden*	"desliga-se" do grupo seleccionado e passa a ser visto pelos restantes membros como estando no estado de *offline*

Tabela 6.3: Opções para modificação de estado do utilizador num grupo

Ao seleccionar qualquer destas opções o utilizador, embora continue a pertencer ao grupo, altera a forma como as interacções com o mesmo são geridas. Note-se que, as comunicações efectuadas quando o utilizador se encontra em modo *block*, não são recuperadas, ou seja, ao deixar de estar em modo *block*, as mensagens que entretanto lhe foram dirigidas, não lhe serão entregues.

Os restantes comandos e funcionalidades para a manipulação dos grupos são idênticos aos anteriormente referidos para a interface de grupos explícitos, com o acesso ao contexto, conversação e configuração/manipulação de ficheiros, acessíveis através dos mesmos três *icons*, presentes no canto inferior esquerdo da interface. Não tem no entanto permissões para remover, criar ou alterar a sua filiação nestes grupos. Pode no entanto observar quais os grupos de sistema (implícitos) actualmente definidos no sistema.

User Profile (dados e preferências)

A interface de *user profile* (figura 6.12) dá acesso a um painel de preferências onde o utilizador pode preencher os seus dados pessoais e seleccionar os seus interesses. É a conjugação desta informação, explicitada pelo utilizador, com os conjuntos de regras definidas pela "administração" do sistema, que colocam um utilizador como membro de um grupo implícito, ou seja, definem a constituição deste tipo de grupos.

Note-se que esta informação de perfil de utilizador, pode também ser utilizada noutros contextos, para além do dos grupos implícitos, sendo dependente dos objectivos definidos pela aplicação.

Os dados podem ser modificados em qualquer momento pelo utilizador, reflectindo-se essas alterações de forma dinâmica, em particular no que se refere à constituição dos grupos implícitos existentes. Ao confirmar a modificação de dados do utilizador (*update*) é desencadeado pela aplicação o processo de actualização de dados do utilizador, através da actualização dos dados no sistema de informação e da chamada da função de $updateProfile()$ do servidor de grupos implícitos.

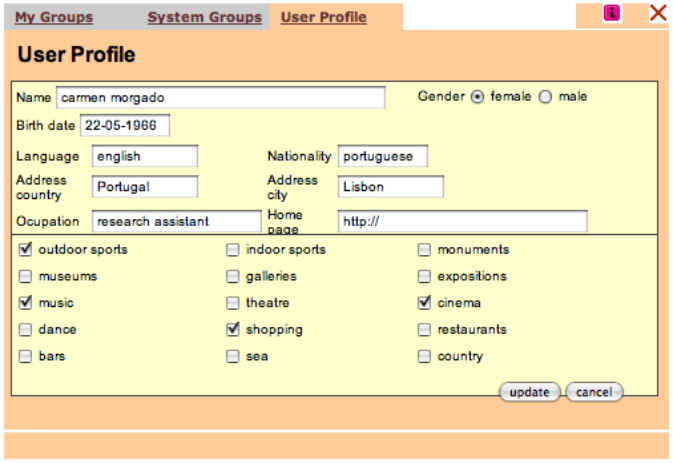

Figura 6.12: Menu de configuração do perfil e dados do utilizador (*User Profile*)

6.4.2 Descrição de funcionalidades de administração

Ao nível de administração do sistema, foi desenvolvida uma aplicação que permite, através de uma interface gráfica simples (apresentada na figura 6.13), desenvolver algumas tarefas de gestão e análise das actividades desenvolvidas no sistema. É também através desta aplicação de administração que se tem acesso ao menu onde são definidas as regras que dão origem à criação dos grupos implícitos[7].

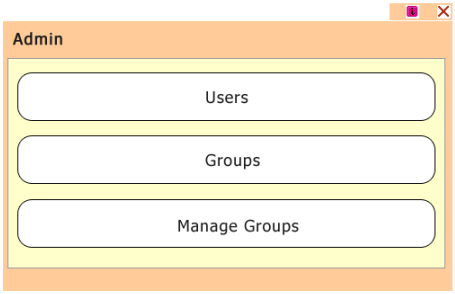

Figura 6.13: Interface de administração

O administrador de sistema é um tipo de utilizador que tem acesso a uma interface que lhe permite analisar o que se passa no sistema ao nível do número de utilizadores registados e activos (opção *Users*), dos grupos explícitos e implícitos existentes (opção *Groups*) e da criação de novos grupos (opção *Manage Groups*).

A última opção referida, desencadeia a abertura de um novo menu onde são pedidos, ao utilizador, os dados necessários para a criação do grupo. Os dados pedidos referem-se ao nome que deseja atribuir ao grupo, ao tipo de grupo e à configuração e permissões que deseja definir para o grupo. No caso particular da criação de um grupo implícito, é apresentado um menu com o conjunto de atributos que podem ser

[7]No desenvolvimento deste protótipo, foi considerado que somente através desta aplicação de administração seria possível criar grupos implícitos.

seleccionados e com base nessa selecção, são filiados automaticamente no grupo, os membros que possuem esses atributos. Este menu é semelhante ao apresentado ao utilizador para este especificar as suas características e preferências (*User Profile*).

6.4.3 Realização do protótipo

Após a descrição das funcionalidades básicas da aplicação (do ponto de vista do cliente), é apresentada a sua arquitectura e exemplificada a forma como foi utilizado o modelo proposto no seu desenvolvimento.

São também descritos alguns aspectos relacionados com a forma como se processou à concepção e construção da interface Web da aplicação. Como já foi referido, as características inerentes às tecnologias Web no cliente (HTML, CSS, Javascript), associadas ao facto de se ter assumido que a implementação no cliente seguiria o mais possível as metodologias tradicionais de desenvolvimento Web, levantaram algumas questões de implementação.

Dado que a proposta do modelo de grupos foi desenvolvida em Java, pressupõe-se que cada cliente tem por base um objecto cliente da classe *MagoEntity*, que lhe permite ter acesso ao conjunto das funcionalidades disponibilizadas pelo modelo.

A arquitectura do protótipo desenvolvido tem por base um conjunto de servidores que têm a seu cargo as tarefas de gestão de grupos e de dados, os quais são acedidos por clientes (utilizadores), através de diferentes tipos de dispositivos, como pode ser observado na figura 6.14.

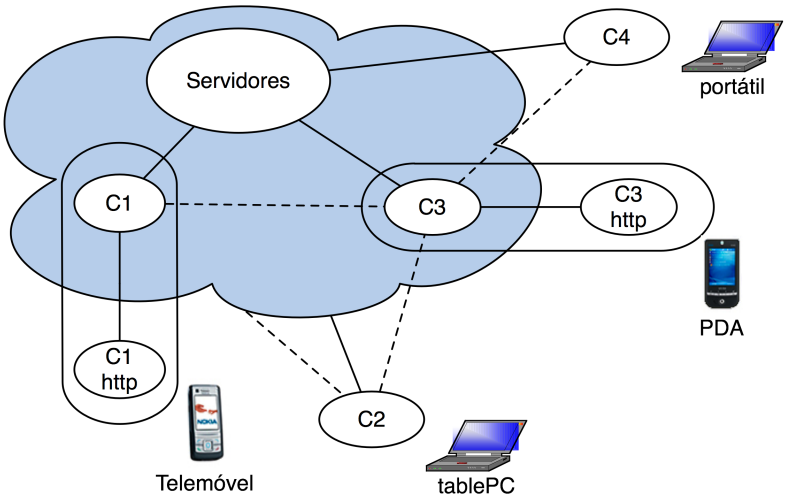

Figura 6.14: Arquitectura geral do protótipo

Esses clientes podem ser executados nos dispositivos locais ou podem ser configurados de forma a obedecerem a uma estrutura como a referida anteriormente, tendo clientes simples e clientes compostos para os dispositivos com menor capacidade computacional.

Tornou-se também necessário integrar a interface com a arquitectura de suporte do modelo, desenvolvida sobre uma plataforma Java. Essa integração levou à criação de um elemento intermédio (*relay server*) responsável por servir, através de um canal http, os cliente Web e simultaneamente, integrar os objectos Java (JSP), responsáveis pela ligação com os restantes elementos da arquitectura.

De seguida, é apresentada uma descrição sumária da arquitectura adoptada para a interface Web, sendo posteriormente descritos alguns dos detalhes associados ao desenvolvimento da própria interface.

Desenvolvimento da interface Web

A utilização de clientes Web no sistema, considerada fundamental dada a generalização da utilização do Web browser como meio de interface com o utilizador, tem por base uma organização clássica que retira do cliente (browser) parte da computação, a ser realizada no cliente.

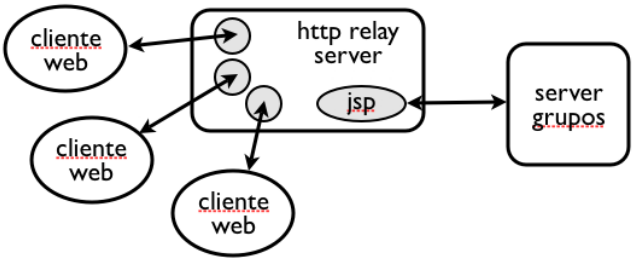

Figura 6.15: Arquitectura para web-support

O http *relay server* suporta as necessidades computacionais específicas de cada cliente do sistema, relegando para o cliente Web um papel mais restrito ao nível da gestão da interface com o utilizador (figura 6.15).

Para cada um dos clientes, o servidor disponibiliza por protocolo HTTP, toda a informação necessária à gestão da interface com o utilizador.

A ligação entre o browser e este servidor que disponibiliza, por HTTP, a actualização dos conteúdos, é em tudo idêntica à de uma qualquer ligação a um servidor Web [Fou06], não fossem algumas particularidades do funcionamento da aplicação, que colocaram alguns desafios ao desenho do cliente web.

Concepção do cliente Web

Nesta breve descrição dos detalhes de implementação do cliente Web, são focados os dois aspectos mais relevantes da sua implementação que, como já foi previamente referido, se situaram ao nível do desenho gráfico da interface e da actualização dos dados.

Em rigor, tratam-se de duas faces de um mesmo problema: a necessidade de, sem uma acção explícita de refrescamento por parte do utilizador (o tradicional *reload* do browser), os dados contidos na interface poderem ser alterados por acção de um outro cliente.

Foi tomada a opção de desenhar a interface com base em CSS [Hol03, FF06], pelo facto de permitir um posicionamento absoluto e facilitar a manipulação de *layers* e ocultação de objectos.

Seguidamente, procurou-se uma solução que permitisse a maior separação possível entre os elementos de gráficos ou visuais e o código associado à sua consulta e actualização. A interface recorre à linguagem Javascript [HP02] para garantir uma actualização da interface com base nos dados presentes em estruturas de dados mantidas na página.

Isto significa que a alteração visual de um elemento na página, por exemplo um novo membro, pode ser efectuada inserindo os dados relativos a esse elemento numa estrutura de dados interna à página Web, sendo garantido que, num espaço de tempo mínimo, essa actualização da estrutura será visível no *browser*.

Esta solução resolveu parte do problema: todos os elementos da interface poderiam ser actualizados dinâmicamente sem intervenção do utilizador, pela simples actualização da estrutura de dados interna à página.

A outra parte do problema prende-se com a necessidade de actualizar essa estrutura de dados, interna à página, com dados do servidor, sem ser necessário proceder a um recarregamento integral da página.

Para resolver esta questão adoptou-se a tecnologia Ajax [UD07], que permite que, de uma forma assíncrona, dados provenientes de um servidor HTTP alimentem um página Web (figura 6.16), existindo do lado da página, código Javascript que responde a eventos gerados em cada actualização e no caso particular desta aplicação, actualizem a sua estrutura de dados interna.

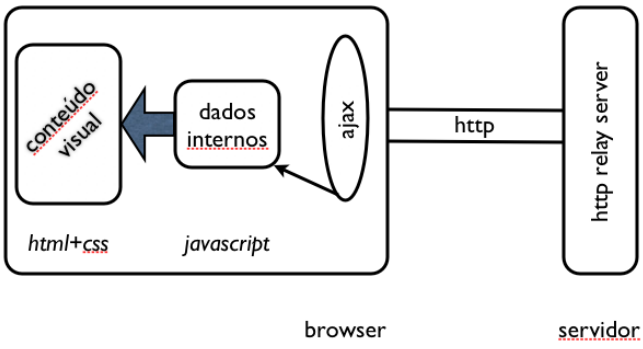

Figura 6.16: Arquitectura do lado do cliente

Estas necessidades são específicas da interface Web, não sendo necessária a sua implementação no caso dos clientes Java ou em outros clientes que possuam capaci-

dade computacional directa no cliente. No entanto, a sua adopção pode ser encarada em qualquer situação em que se pretenda libertar o cliente de algum peso computacional ou, muito simplesmente, disponibilizar clientes leves que não introduzam uma carga excessiva no ambiente em que são executados. Isto poderá ser particularmente sensível em casos que envolvam dispositivos móveis, normalmente com capacidade computacional limitada.

Sistema de informação

Neste protótipo, foi utilizada como plataforma de suporte ao sistema de informação, o ORACLE Database 10g Express Edition [McL07]. O modelo de dados foi definido tendo por base os requisitos da aplicação, que neste caso envolvia dados relacionados com as propriedades e preferências dos utilizadores, bem como as características definidas para os grupos.

Neste caso, foi utilizado o mesmo suporte de informação para a gestão dos dados próprios da aplicação e dos dados próprios do modelo (definidos no capítulo 5). Foi definido, à semelhança do realizado para o modelo, um conjunto de primitivas base que modelam as interacções existentes entre a aplicação e o sistema de informação. Essa primitivas possibilitam, por exemplo, inserir um novo utilizador (com o seu conjunto de preferências e propriedades), consultar, alterar e removê-lo do sistema.

6.5 Exemplos de integração em aplicações

Como já foi referido no capítulo inicial (capítulo 2) existe todo um conjunto de aplicações que tem vindo a surgir, cada vez mais com necessidades ao nível das interacções, mobilidade e organização, que poderão tirar partido da utilização do modelo aqui proposto, alargando assim o leque de funcionalidades disponibilizadas aos seus utilizadores.

Aqui, vão ser analisadas algumas das aplicações, que poderiam ser modificadas com o modelo proposto e a forma como as novas funcionalidades poderiam ser integradas, sendo apresentados alguns exemplos concretos da utilização dos padrões de organização de grupos e de comunicações propostos no modelo.

Em particular, é analisada a possibilidade e a forma de integração do modelo em projectos de apoio ao ensino [CC07] e no projecto InStory [CAC+05]. Discute-se também o modo como a introdução da abstracção de grupos na organização permite a criação de um novo tipo de funcionalidades, bem como simplificar a definição de alguns dos já existentes. Os exemplos apresentados correspondem a dois tipos de utilização bastantes diferentes embora de um modo geral as interacções presentes em ambos sejam, de um modo geral, semelhantes. No primeiro caso, são apresentados exemplos da formação e estruturação de comunicações de grupo, sendo, no segundo caso, dada

uma maior ênfase à utilização do espaço de partilha, como forma de colaboração e interacção em grupos.

6.5.1 Sistema de apoio ao ensino

Já foram anteriormente referidos diversos tipos de aplicações de apoio ao ensino (capítulo 2), tendo sido também analisado um cenário de aplicação possível, onde foram descritas diversas utilizações de padrões de organização de grupos, aplicadas a um domínio de aplicação relacionada com o ensino (*campus* universitário).

Aqui é referido um exemplo de utilização especifico tendo por base uma aplicação em desenvolvimento, inserida no contexto do projecto VideoStore [CC06]. Este sistema permite tornar acessível um conjunto de materiais de apoio a aulas, assim como um conjunto de funcionalidades que permitem aos alunos aceder e editar as suas anotações referentes aos conteúdos disponibilizados. Os utilizadores dispõem de uma interface (figura 6.17) que lhes permite visualizar e manipular vídeos de aulas, anotações vídeo e *slides*, podendo os alunos efectuar as suas próprias anotações sobre os vídeos disponibilizados.

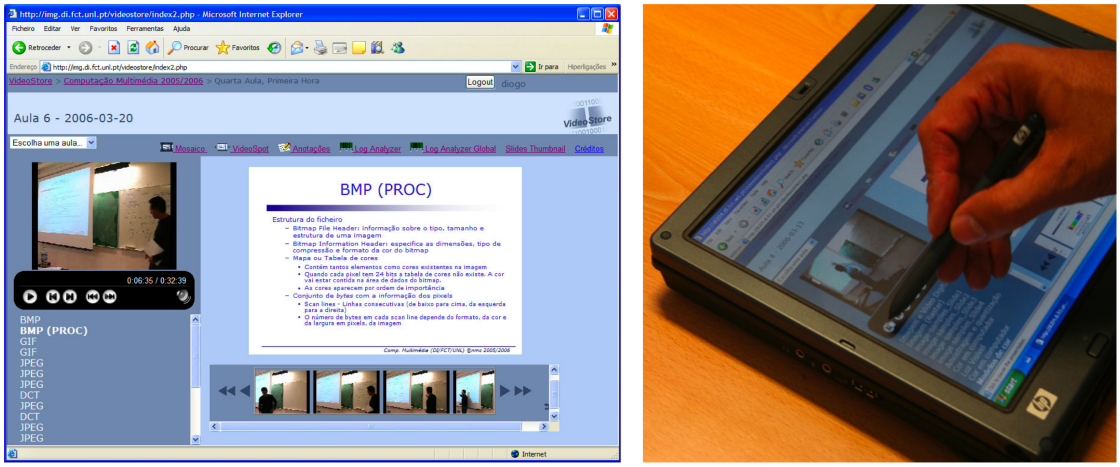

Figura 6.17: Aplicação VideoStore, com interface desenvolvida para browser

O sistema actual possui uma arquitectura baseada num servidor de conteúdos, que podem ser consultados por elementos registados no sistema [CC07]. A alteração dos conteúdos, assim como a sua remoção, só pode ser efectuada pelo produtor dos mesmos. Por exemplo, os vídeos disponibilizados só podem ser modificados ou removidos pelo seu produtor e as anotações efectuadas sobre esses vídeos ficam associadas ao vídeo mas não produzem alterações no vídeo original. As anotações podem ser visualizadas e modificadas pelos seus produtores ou por um conjunto de elementos especificado por este. As anotações são, de um modo geral, conteúdos (texto, ou imagem) que estão associados a um vídeo[8].

[8]Embora, de um modo geral, as anotações estejam associadas a um vídeo podem também ser efectu-

A integração de um modelo de abstracção de grupos, neste tipo de aplicação, permite a definição de estruturas de organização que melhor capturam as interacções existentes dentro de uma turma (sala de aula). Um dos exemplos possíveis de utilização é a definição de um grupo "turma" através do qual o professor disponibiliza conteúdos, como por exemplo vídeos, textos de apoio, ou algum tipo de aviso. Qualquer aluno "inscrito" na turma, ou seja, membro do grupo "turma", passa a ter acesso aos conteúdos produzidos dentro do contexto da turma, não sendo estes materiais visíveis fora deste espaço.

Os alunos podem criar os seus próprios grupos, que podem ser utilizados como "locais" de colaboração e cooperação para a realização de trabalhos. Estes grupos podem ser utilizados como um espaço, onde os alunos partilham informação, colaboram e comunicam entre si, para realizarem os seus trabalhos.

A arquitectura base do sistema inicial é composta por três elementos principais:

- uma base de dados com os conteúdos manipulados (*slides*, vídeos e anotações);
- um servidor responsável pela gestão de acessos aos dados;
- uma interface através da qual os clientes podem aceder aos dados.

A integração do sistema existente com o modelo de grupos proposto, produz as alterações que podem ser visualizadas na figura 6.18. Como se pode observar, esta integração traduz-se essencialmente em dois pontos:

- alteração do suporte de dados de modo a este acomodar a informação referente aos utilizadores (atributos dos alunos e professores) e aos grupos;
- introdução do componente MAGO e de uma interface que lhe permita aceder ao sistema de informação (database).

O componente MAGO e a interface DB, dão acesso às funcionalidades de comunicação e de gestão de grupos e de configuração e criação de grupos implícitos com base nos perfis de utilizadores.

Cada elemento do sistema (alunos e docentes) passa a ser modelado como entidade elementar, passando a ter acesso a todo o conjunto de primitivas do modelo o que lhe permitem explorar as funcionalidades de grupos tendo acesso aos mecanismos de comunicação definidos no modelo proposto. Os alunos e os professores passam a ser entidades elementares, tendo definida, para cada uma, uma chave de acesso no sistema, criada quando do registo. Esta identificação é necessária sempre que uma destas entidades reentra no sistema, ou seja, sempre que inicia uma sessão. A criação de novos grupos pode ser desencadeada pelos diversos intervenientes no sistema, por exemplo:

adas sobre qualquer dos conteúdos.

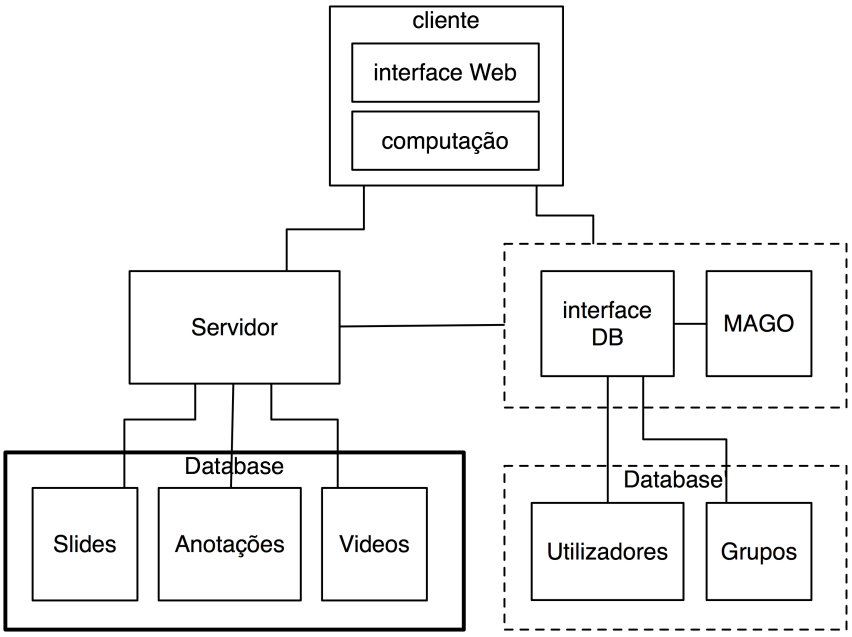

Figura 6.18: Arquitectura VideoStore integrada com o modelo proposto

- professor: cria grupo explícito "turma", especificando uma lista de participantes e as suas chaves de acesso;
- aluno: cria grupo explícito, por exemplo "grupo de estudo", para colaboração e partilha de informação e onde a admissão de novos membros deve ser validada por ele;
- aluno: define conjunto de características, para a criação de grupo implícito dos potenciais elementos que indicaram ter interesse no tema "animação por computador".

A organização com base em grupos de interesses (explícitos ou implícitos), permite uma partilha de conteúdos e comunicação mais selectiva e específica entre os elementos intervenientes. Cada elemento activo do sistema é representado como uma entidade elementar passando, desta forma, a ter acesso a todas as funcionalidades definidas no modelo para estas entidades. Para dar suporte a esta nova estrutura de interacções, têm que ser introduzidas alterações ao nível das interfaces, de modo a dar acesso à manipulação do novo conjunto de funcionalidades que naturalmente surge. Também ao nível do sistema de informação que dá suporte à aplicação já existente, têm que ser introduzidas estruturas, relacionadas com as novas características que podem ser definidas para os elementos, assim como as associadas à gestão do modelo de grupos.

De seguida, são exemplificadas formas de aplicação de alguns dos padrões de utilização do modelo.

Criação de uma turma por parte de um professor

Um professor é representado por uma entidade elementar do sistema (MagoEntity), sendo assumido que já se encontra registado (possui um identificador único de acesso) e a aplicação garante que tem permissão para criação deste tipo de grupo.

```
...
// professor é uma entidade elementar, que deve ter previamente efectuado
// a instanciação de um objecto da classe MagoEntity
mago_prof = new MagoEntity(...);

// e ter efectuado o seu registo onde passa o seu login e pwd
// tendo-lhe sido atribuído um identificador (id_prof)
id_prof = mago_prof.register(...)

// e criado uma entrada no sistema de informação
si_prof = new AppSIType(...)
sip = si_prof.newEntity(...)
// e actualizado os seus dados
si_prof.setEntity(...)
...
// criação da turma
lista = { cria lista de membros admissíveis no grupo,
                com a estrutura [nº aluno, chave de acesso] }
max = sizeOf( lista );
nome_grupo = { define um nome válido para a nova turma }

// cria entrada do grupo no SI
si_prof.si_newGroup( nome_grupo, login; out:res );

if ( res )
        // criação de entrada do grupo no Si efectuada com sucesso
        ...
        valida = mago_prof.create( id_prof, nome_grupo, EXPLICIT, BY_LIST, max, lista;
             out: id_grupo );
        if ( valida )
                //  criação de turma efectuada com sucesso
                ...
                // actualiza dados no SI
                lista_params = [ id_prof, explicit, by_list, lista, date, info ];
                si_prof.si_setGroup( nome_grupo, id_grupo, list_params );
                ...
                // professor pode proceder à sua filiação
                info = [ nº professor, chave de acesso para admissão no grupo ]
                mago_prof.join( id_prof, id_grupo, info; out: prof_turma_id);

                // actualização da informação no SI, referente à constituição do grupo
                si_prof.si_putGroupMember( id_grupo, id_prof; out: res2 );
                if ( res2 )
                        // OK
                else
                        // tratamento de erro referente à actualização do grupo no SI
        else
                // tratamento de erro de criação de grupo
else
        // tratamento de erro de criação de entrada de grupo no SI
...
```

Listagem 6.4: Pseudo código de criação de um grupo "turma" por parte de um professor

Após executar esta sequência de acções, é criado um novo grupo, que passa a ser visível por todos os membros e no qual os alunos autorizados se podem inscrever (ou seja filiar). Nesta situação, a filiação de um aluno (registado) no grupo "turma" recém criado, só é efectuada se o aluno pertencer à lista definida pelo criador do grupo.

```
...
// aluno é uma entidade elementar (mago_stu), e deve ter efectuado
// o seu registo no sistema onde lhe foi atribuído um identificador único (id_stu)
// e foi efectuada a actualização dos seus dados no SI (siStu)

...
// preenche informação necessária para admissão ao grupo
info = [ nº aluno,  chave para admissão ao grupo ];

// selecciona turma onde se pretende inscrever e obtém o identifidador
getGroupId( nome_grupo, id_grupo );

// filia-se na turma
valida = mago_stu.join( id_stu, id_grupo, info; out: stu_turma );

if ( valida )
        // procede à actualização do sistema de informação
        siStu.si_putGroupMember( id_grupo, id_stu; out: res );

        if ( res )
                // OK
        else
                // processa erro de actualização de constituição do grupo no SI
else
        // processa erro de filiação no grupo
...
```

Listagem 6.5: Pseudo código da filiação de um aluno no grupo criado pelo professor

A partir do momento em que um aluno se torna membro do grupo "turma", passa a ter acesso ao conjunto de conteúdos e materiais produzidos no contexto desse grupo. Não é possível definir qualquer tipo de restrição, ao nível do modelo, relativamente aos membros que podem produzir conteúdos para o grupo "turma", sendo da responsabilidade da aplicação a criação de restrições ao nível das funcionalidades a que cada entidade tem acesso.

Criação de grupo de estudo e partilha de conteúdos

Os alunos são entidades elementares do sistema que, de acordo com as permissões definidas pela aplicação, podem criar grupos, dentro dos quais podem estabelecer diversas formas de interacção. No exemplo aqui apresentado, é desencadeada a acção de criação de um grupo de estudo por parte de um aluno, o qual fica também responsável por gerir os pedidos de acesso ao grupo, por parte dos alunos que pretendam filiar-se.

```
...
// aluno deve ser uma entidade elementar, e deve ter efectuado
// o seu registo no sistema (mago_stu), tendo-lhe sido
// atribuído um identificador único (id_stu)
// e foi efectuada a actualização dos seus dados no SI (siStu)
...

// criação de grupo de trabalho
max = 5;
nome_grupo = { define um nome válido para o grupo }

// cria entrada do grupo no SI
siStu.si_newGroup( nome_grupo, id_stu; out:res );

if ( res )

        // cria grupo
        valida = mago_stu.create( id_stu, nome_grupo, EXPLICIT, BY_CREATOR, max; out:
            id_grupo );

        if ( valida )
                // actualização da informação do grupo no SI
                // onde são colocados parâmetros definidos pela aplicação
                lista_params = [ id_stu, explicit, by_creator, grupoTrabalho, date, "
                    Grupo de trabalho da disciplina de Computação"  ];
                siStu_setGroup( id_grupo, nome_grupo, list_params; out: res2 );

                if ( res2 )
                        //  criação de grupo efectuado com sucesso procede à sua
                            filiação
                        // como se trata do próprio criador do grupo a admissão é
                            directa

                        valida2 = mago_stu.join( id_stu, id_grupo, info; out:
                            stu_Grupo );
                        if ( valida2)
                                // filliação efectuada com sucesso
                                // actualiza constituição do grupo no SI
                                siStu.si_putGroupMember( id_grupo, id_stu; out: res );
                                if ( res3 )
                                        // OK
                                else
                                        // trata erro de actualização da
                                        // constituição do grupo no SI
                        else
                                // trata erro de filiação
                else
                        // trata erro de actualização de dados do grupo no SI
        else
                // trata erro de criação do grupo
else
        // trata erro de criação de entrada no SI do novo grupo
                ...
```

Listagem 6.6: Pseudo código de criação de um grupo de trabalho por parte de um aluno

Ao executar a sequência de acções descritas, passa a existir um novo grupo sobre o qual os alunos, que assim o pretendam, podem efectuar pedidos de filiação.

```
...
// aluno é uma entidade elementar, e deve ter efectuado
// o seu registo (mago_stu1) no sistema onde lhe
// foi atribuído um identificador único (id_stu1)
// e foi efectuada a actualização dos seus dados no SI (siStu1)

// preenche informação necessária para admissão ao grupo
// a mensagem é emitida para todos os membros do grupo
nome =  "Pedro Silva";
mensagem = "Olá posso fazer parte deste grupo de trabalho?";
info = [ nº aluno, nome, mensagem ];

// selecciona turma onde se pretende inscrever e obtém o identifidador
getGroupId( nome_grupo, id_grupo );

valida = mago_stu1.join( id_stu1, id_grupo, info; stu1_Grupo );

if ( valida )
        // se filiação aceite
        // procede à actualização do sistema de informação
        siStu1.si_putGroupMember( id_grupo, id_stu1; out: res );
        if ( res )
                // OK
        else
                // trata erro de actualização da constituição do grupo no SI
else
        // não foi admitido no grupo
...
```

Listagem 6.7: Pseudo código do pedido de filiação de um aluno no grupo de trabalho criado

O pedido de filiação, por parte dos candidatos a membros, é tratado ao nível da entidade responsável pela criação do grupo, sendo enviada, quando da chamada de *join*(), uma mensagem ao criador, com o pedido de filiação. Este, ao receber o pedido, efectua o seu processamento que se reflecte, ao nível da aplicação, por uma sinalização na interface de pedido de entrada a que a entidade criadora deve dar resposta. A sua resposta (positiva) valida a entrada do candidato a membro no grupo. A modificação da composição do grupo não desencadeia, ao nível do modelo, nenhum processo de notificação, sendo da responsabilidade da aplicação efectuar a actualização dos dados ao nível da interface cliente.

Estes grupos podem também ser utilizados para suportar a partilha de conteúdos entre os membros. Por exemplo, um dos alunos escreveu um texto que pretende partilhar com o seu grupo de estudo e receber os comentários com as opiniões dos restantes membros.

```
...
// informação referentes aos dados (ficheiro)
file_type = "documento"         // tipo de ficheiro
comment = "em revisão"          // comentários informativos inseridos pelo aluno
file_name = "report.doc"        // ficheiro com relatório (report.doc)
date = getActualDate();
```

```
// cria objecto de dados para ser colocado no SI
data = new AppContextData( file_type, comment, file_name, date );

// actualiza os seus dados
siStu.si_updateCEntity( id_stu, data; data_id );

// actualiza os dados do grupo no SI
siStu.si_updateCGroup( id_grupo, id_stu, data; out: data_ref );


if ( data_ref )
        // actualização efectuada com sucesso

        // cria objecto lista com dados da aplicação para o espaço partilhado
        list = new AppData( file_type, comment, file_name, date, ref_data);

        // cria objecto tuplo com dados para o espaço partilhado
        // com list e data_ref, que permite acesso aos dados no SI
        dataT = new TupleDataType( list + data_ref);

        // actualiza os dados no espaço partilhado do grupo
        // desencadeando notificação do membros do grupo
        valida = mago_stu.update( id_stu, id_grupo, PUBLIC, NOTIFY_UPD, dataT );

        if ( valida )
                // actualização efectuada com sucesso
        else
                // processa erro de actualização do espaço partilhado do grupo
else
        // processa erro de actualização de dados no SI
...
```

Listagem 6.8: Pseudo código da colocação para partilha, de ficheiro por parte de um aluno do grupo

O tratamento da notificação por parte dos membros do grupo desencadeia um processo de actualização da informação visível na interface definida pelo programador da aplicação. De um modo geral, essa alteração pode passar por colocar o nome do grupo onde se deu a actualização noutra cor (ou a piscar) actualizando de seguida a lista de conteúdos partilhados por esse grupo. Ao seleccionar esse conteúdo, o aluno pode optar por efectuar a leitura desses dados, podendo de seguida proceder à visualização do ficheiro com o relatório.

No pseudo-código apresentado de seguida, um dos alunos do grupo efectua a leitura do ficheiro, comunica directamente com o autor do mesmo e envia para o grupo (difunde) uma mensagem com um resumo dos seus comentários, colocando seguidamente a versão completa dos mesmo no espaço de partilha do grupo, que assim ficam disponíveis para serem consultados pelos restantes membros.

Ao efectuar a actualização de dados com a primitiva de *update*() com opção de NOTIFY, é publicado um evento de UPDATE que, como já foi referido, desencadeia em todos os membros activos do grupo, a chamada de um método de atendimento. Este é configurado pela aplicação que pode, por exemplo, definir que seja desencadeada

uma simples actualização da informação visível ao aluno (utilizador) ou uma nota de aviso de novos dados para consulta.

```
...
// recebeu notificação de actualização com uma identificação dos dados

// le informação do espaço do grupo
// pode consultar directamente o SI, se já possui todos os dados para acesso
// v = siStu.si_getCGroup( id_grupo1, ref_data; out: data_obj );
...

// ou ler espaço partilhado para obtenção da referencia aos dados no SI e desta forma
    aceder ao ficheiro
// definição do objecto de pesquisa
dataT = new TupleDataType( ... + data_id );

v = mago_stu1.consult( id_stu1, id_grupo1, id_stu, -, NO_BLOCK, dataT; out: dataT );

// extrai informação da data lida do espaço dataT
ref_data = dataT.getRef( );
val = siStu.si_getCGroup( id_grupo1, ref_data; out: data_obj );

// visualiza ou faz download do ficheiro report.doc para posterior leitura do mesmo

// envia mensagem ao membro produtor a avisar que irá enviar comentários para o grupo
// emissor stu1, receptor stu produtor do report
v =  mago_stu1.send(id_stu1, id_stu, PASSIVE, "Vou ler e envio os meus comentários
    para o grupo, asp" );

...
// coloca report_STU1 para partilha no grupo
//
...

// mensagem a difundir para o grupo id_grupo1
msg = "Report ok, vejam ficheiro com pequenas correcções report_STU1.doc";

v =  mago_stu1.send(id_stu1, id_grupo1, SPREAD, msg );
...
```

Listagem 6.9: Pseudo código de consulta de dados do espaço partilhado por parte de um aluno do grupo que recebeu a notificação

Ao efectuar-se uma actualização de dados no espaço partilhado do grupo sem notificação, tal significa que esta só será observada pelos membros do grupo, quando for desencadeada, por parte de cada membro do grupo, uma operação de consulta de dados do grupo.

Criação de grupo implícito com base em características das entidades

Neste exemplo de utilização, um aluno cria um grupo, tendo por base determinada característica que deve ser observada por todos os membros. O grupo não é criado por uma acção explícita dos seus membros, mas devido à especificação de propriedade(s) que devem ser observadas pelas entidades para pertencerem ao grupo. No caso apresentado, foi utilizada uma propriedade/atributo "interesse" com o valor "animação

por computador". Todas as entidades do sistema que possuam um atributo com este valor são automaticamente filiadas no grupo. Os novos elementos, que entrem posteriormente no sistema, passam também a fazer parte do grupo, se satisfizerem essa propriedade[9].

```
...
// constrói lista de propriedades e respectivos valores
list_params = [ tema1, "animação por computador" ];

// atribui um nome ao grupo
nome_grupo =  { define um nome válido para o novo grupo }

// cria entrada do grupo no SI
si_prof.si_newGroup(nome_grupo, id_prof; out: res );

if ( res )
        // criação de entrada no SI do grupo efectuada com sucesso
        valida = mago_prof.create( id_prof, nome_grupo, IMPLICIT, ALL, -1, list_params
            ; out: id_grupoI );
        if ( valida )
                // criação do grupo efectuada com sucesso
                list = [ id_prof, implicit, list_params ]
                si_prof.si_setGroup( nome_grupo, id_grupoI, list; out: res2 );
                if ( res2 )
                        // actualização dos dados do grupo ok
                else
                        // trata erro de actualização dos dados do grupo
        else
                // trata erro de criação de novo grupo implicito
else
        // trata erro de criação de entrada no SI

...

// Após a criação do novo grupo pode ser difundir informação    para o grupo
msg = "Seminário sobre edição de video dia 20 Setembro às 14:00 sala 1.1. Entrada
    livre";
v =  mago_stu1.send(id_prof, id_grupoI, NOTIFY, msg );


// Assumido que entidade que criou o grupo pertence a este grupo,
// de modo a ter acesso aos mecanismos de comunicação.
// Pode utilizar o espaço partilhado do grupo de modo a comunicar
// ou partilhar dados.
type = "mensagem"
comment = "Seminário sobre edição de video dia 20 Setembro às 14:00 sala 1.1. Entrada
    livre";

// cria objecto de dados para ser colocado no SI
data = new AppContextData( type, comment, date );

// actualiza os dados do grupo no SI
sinfo.si_updateCGroup( id_grupoI, id_stu, data; out: ref_data );

// cria objecto lista com dados da aplicação para o espaço partilhado
```

[9]Se a entidade modificar o valor desta ou de outras propriedades, é desencadeado o processo de "reavaliação" da filiação dessa entidade.

```
list = new AppData( file_type, comment, file_name, date, ref_data);

dataT = new TupleDataType(list + ref_data);

// actualiza os dados no espaço partilhado do grupo
v = mago_stu.update( id_stu, id_grupoI, PUBLIC, NO_NOTIFY_UPD, dataT );
...
```

Listagem 6.10: Pseudo código de criação de um grupo implícito

As entidades que se encontram registadas no sistema e que observem esta propriedade com este valor, são automaticamente filiadas no grupo, quando da criação deste. No processo de registo da entidade, é efectuada uma avaliação das características da entidade e dos grupos implícitos actualmente existentes, de modo a desencadear o processo de actualização das filiações da entidade e da constituição dos grupos. O mesmo se passa quando é efectuada uma modificação das características da entidade, sendo desencadeado um processo de reavaliação da filiação da entidade nos grupos implícitos actualmente existentes no sistema.

6.5.2 Projecto InStory

O principal objectivo do projecto InStory [CAC+05] consiste na definição e implementação de uma plataforma móvel, que permita desenvolver um conjunto de aplicações passíveis de interesse num espaço turístico. O espaço de implementação escolhido para este projecto foi a Quinta da Regaleira, um complexo arquitectónico situado em Sintra que inclui um palacete, do início do século XX, uma pequena capela e um jardim com diversos apontamentos de interesse. Existem, ao longo do jardim, galerias com labirintos, grutas, fontes, estátuas e um poço seco, sendo este último um dos pontos mais distintivos do complexo.

Figura 6.19: Menus da aplicação já desenvolvida, referentes ao tipo de informação que pode ser obtida e a mapa de apoio à navegação [CAC+05]

Todo este complexo oferece um espaço privilegiado para a construção de narrativas, de jogos e de acesso interactivo a informação de um modo geral, por parte de

múltiplos utilizadores móveis.

Ao longo do projecto, têm vindo a ser desenvolvidas aplicações na área de apoio à "navegação" e de jogos, que permitam a um visitante obter dados e informações úteis, sobre o local onde se encontra, de forma simples de dinâmica, com base em dispositivos móveis, PDAs em particular. Na figura 6.19 podem ser observados exemplos de alguns dos menus existentes na aplicação desenvolvida especificamente para PDAs. Actualmente estão também a ser desenvolvidas aplicações de partilha e pesquisa de conteúdos multimédia, em particular fotos, captadas pelos diversos visitantes [RMJDFC07].

Em termos de infraestrutura física de suporte, a quinta possui cobertura de rede (sem fios) e tem também instalado um sistema de localização (Ekahau [Eka07]), que permite a detecção, em qualquer ponto, do posicionamento actual do visitante. Paralelamente e no caso dos dispositivos possuirem GPS (Global Positioning System), possui também uma forma de lidar com a informação oferecida por estes, conseguindo desta forma determinar a posição ocupada pelos visitantes.

A arquitectura actual do sistema de suporte às diferentes aplicações, tem por base um conjunto de servidores, situados no edifício dos serviços e diversos dispositivos móveis (clientes) transportados pelos visitantes.

Algumas aplicações, desenvolvidas no âmbito deste projecto, estão relacionadas com uma diversidade de aspectos: apoio à navegação, oferecendo aos visitantes informações de acordo com a actual posição ocupada pelo visitante [DJFC07]; definição de jogos e narrativas cujo comportamento evolui de acordo com a posição dos visitantes e das opções por estes tomadas [JDF+07, CAC+05]. No caso das narrativas e jogos, estes desenvolvem-se de acordo com as acções desencadeadas pelos utilizadores, que podem participar activamente no desenrolar dos mesmos, contribuindo para estes, não só com as suas acções e objectos "captados", como também com o seu posicionamento actual e o dos restantes visitantes. Os objectos podem ser objectos virtuais associados a elementos existentes no espaço, como por exemplo estátuas ou edifícios, ou os dados recolhidos pelos próprios visitantes, por exemplo, fotos ou vídeos.

Com base no sistema e infraestrutura existente, foi estudada a utilização de padrões de organização em grupo (explícitos e implícitos), bem como os padrões de interacção combinados que existem no modelo MAGO (comunicação por mensagens, por eventos e por espaço partilhado). A integração da implementação proposta para o modelo, no sistema já existente, permite a definição de novas funcionalidades que podem ser facilmente utilizadas na criação de novos serviços e opções nas aplicações já existentes, bem como no desenvolvimento de novas aplicações que tirem partido das possibilidades oferecidas pelos padrões de organização em grupo disponíveis.

Alguns exemplos de possíveis adaptações de aplicações já existentes:

- nos jogos e narrativas interactivas: a criação de grupos possibilita a definição de novos tipos de jogos onde se tira partido desta forma de estruturação, sendo possível definir por exemplo, padrões de utilização onde as diversas equipas que competem entre si, podem estabelecer formas de colaboração e cooperação entre os seus membros de forma a atingirem um dado objectivo.

- nas visitas guiadas com partilha de conteúdos:

  – a criação de grupos explícitos de visita, que podem ser utilizados como forma dos visitantes partilharem conteúdos e comunicarem entre si. Por exemplo, os membros do grupo de visita podem produzir e disponibilizar, para o seu grupo, fotos que vão tirando ao longo da visita e que podem ser consultadas por todos os membros

  – a criação de grupos implícitos pode ser utilizada como forma de personalizar a informação disponibilizada aos visitantes. Por exemplo, os visitantes de determinada nacionalidade podem ter conteúdos construídos de acordo com o seu perfil. Outro exemplo pode ser a criação de grupos, com base nos visitantes que tiraram fotos num determinado local.

Ao nível da arquitectura do sistema, a integração da implementação do modelo é facilmente conseguida, sendo poucas as modificações introduzidas. Uma das modificações situa-se ao nível do modelo de dados, que deve ser alterado de forma a que seja possível a integração das estrutura de dados que armazenam a informação referente às entidades elementares e aos grupos.

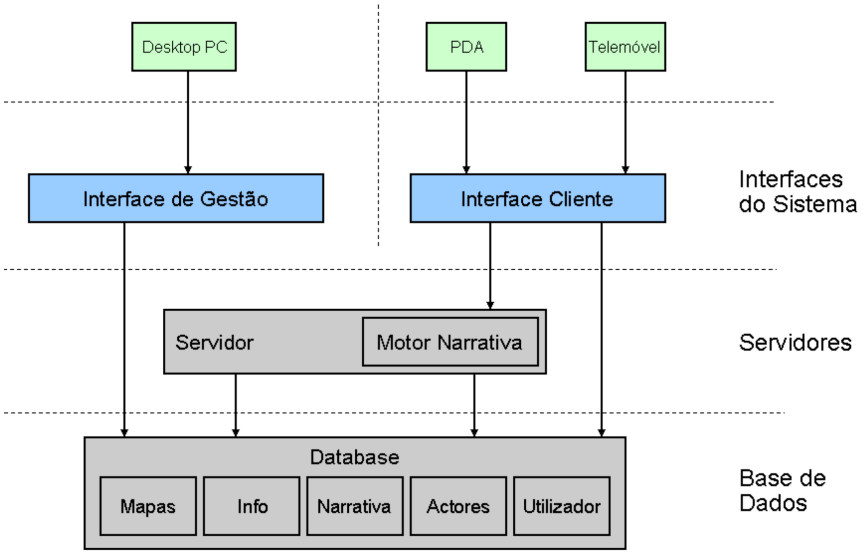

Figura 6.20: Arquitectura de aplicações desenvolvidas no projecto inStory (figura retirada do documento [Dia07])

Partindo da proposta de arquitectura inicial apresentada na figura 6.20, procedeu-se a um conjunto de alterações com vista à integração do modelo MAGO. Essas alterações estão representadas na figura 6.21 e, como se pode observar, reflectem-se na adaptação da base de dados, de forma a acomodar os novos dados referentes aos utilizadores e aos grupos, e na introdução do módulo MAGO e do módulo responsável por gerir e efectuar a interface deste com a base de dados. Este último tem a seu cargo a gestão dos grupos, em particular a criação e manutenção dos grupos implícitos e como tal, tem que interagir directamente com as base de dados existentes.

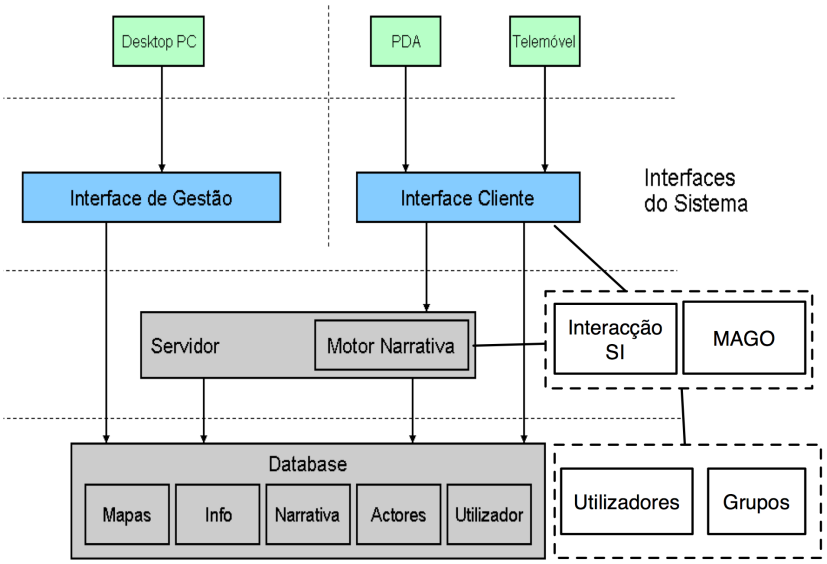

Figura 6.21: Arquitectura modificada com integração dos elementos necessários para utilização do modelo do grupo

Como já foi referido, a utilização do modelo permite simplificar a definição de novas funcionalidades nos jogos e narrativas já existentes, bem como a criação de novos tipos de jogos de modo simples e natural para o programador. Um exemplo simples de jogo, que constitui uma adaptação de um jogo já existente, é o de duas equipas de jogadores que competem entre si na recolha de objectos que se encontram associados a determinados locais. O objectivo do jogo é cada equipa recolher o maior número de objectos, evitando que estes sejam obtidos pela equipa adversária. Os jogadores têm acesso a um sistema de navegação (que lhes indica não só a sua posição actual como a posição dos restantes membros da sua equipa) e a um plano de jogo, que lhes vai dando indicações relativamente à evolução do jogo, assim como possíveis pistas para os passos que devem ser seguidos.

Nesta situação, cada jogador é modelado como uma entidade elementar, sendo criados grupos (com lista de participantes) que representam as equipas. Deve ser definido também um grupo "Jogo" que é composto por todos os participantes no jogo e pelo gestor ("master") do jogo. Este último é o elemento responsável por gerir os "objectos" existentes no local, tem também a possibilidade de enviar informação aos participantes

no jogo (ou a um em particular) e para as várias equipas (figura 6.22).

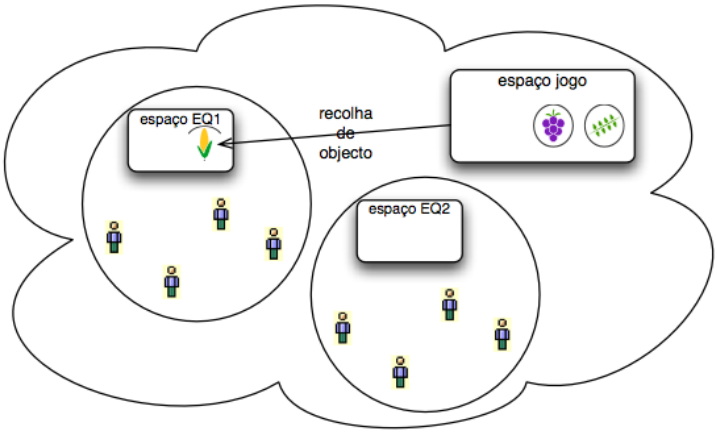

Figura 6.22: Formação de grupos (equipas) num jogo

São apresentadas, de seguida, algumas possíveis utilizações de mecanismos disponibilizados pelo modelo para a realização do jogo sendo, neste caso, dada particular atenção à utilização do espaço de partilha e à comunicação por eventos.

A entidade responsável pela gestão do jogo define os objectos ("virtuais") que existem no jogo. Os objectos são elementos passivos, caracterizados por um tipo e estão directamente associados a um local (ou situação particular do jogo). Estes objectos podem ser modelados sob o formato de tuplos que são colocados, através de *update*(), no espaço de partilha do grupo, a que todos os participantes do jogo têm acesso.

```
...
// criação do "objecto"  por parte da entidade gestora do jogo
// mago_master (com id_master) sendo o grupo do jogo id_jogo1
name = "lança"
type = "artefacto"
property = "matar"
utility = 4
msg_info = "lança que lhe permite lutar com o Leão"
data = new ObjData( name, type, property, utility, msg_info );

// associação do objecto a um local
place = {posição, local ao qual fica associado o "objecto"};
list = new GameObjData( data, place );

// actualização do "objecto" no SI
sinfo.si_updateCGroup( id_jogo1, id_master, list; out: ref_data );

// cria objecto com dados para colocar no espaço do jogo
dataT = new TupleDataType( list + ref_data );

// actualiza os dados no espaço partilhado do grupo
v = mago_master.update( id_master, id_jogo1, PUBLIC, NO_NOTIFY, dataT);
```

```
// actualiza info referente ao local/posição,
// com "colocação" do objecto no local
...
```

Listagem 6.11: Pseudo código de criação de um "objecto" e sua colocação em jogo, por parte de gestor do jogo

Ao chegar a um determinado local, um participante recebe um aviso (notificação) de que existe um objecto que está disponível para recolha, sinalizando este facto, ao nível da interface. Se este optar pela sua recolha, desencadeia uma operação de *get*() do espaço do jogo, ou seja remove objecto do jogo e coloca (*update*()) o objecto no espaço de partilha do grupo. Se, entretanto, outro participante chegar a esse local, já não encontra nenhum objecto disponível.

```
...
// selecciona "objecto", o que permite definir os seus campos
name = "lança";
type = "artefacto";
property = "matar";
utility = 4;
msg_info = "lança que lhe permite lutar com o Leão";
place = {localização actual};

// cria lista com dados para remoção do objecto do espaço do jogo
data = new ObjData( name, type, property, utility, msg_info );
list_params = new GameObjData( data, place );
ref_data =null;
ObjectoD = new TupleDataType(list_param + ref_data);

// desencadeia recolha de objecto do espaço partilhado
v = mago_player1.get( id_player1, id_jogo1, list_params; out: objectoD );

data_id = ObjectoD.getRef(ObjectoD);
sinfo.si_removeCGroup( id_jogo, id_player, data_id; out: res );

// coloca objecto na sua equipa, notificando os restantes membro
sinfo.si_updateCGroup( id_jogo1, id_master, list; out: ref_data );

// actualiza ref do objecto no SI
ObjectoD.setRef(ref_data);
// coloca objecto no espaço da equipa 1
v = mago_player1.update( id_player1, id_equipa1, PUBLIC, NOTIFY, ObjectoD );
...
```

Listagem 6.12: Pseudo código da captura de um "objecto" por parte de um jogador do local onde se encontra

O espaço de partilha também pode ser utilizado pelos membros de uma equipa para fornecerem informação a todos os membros ou como forma de lançarem novas pistas para o jogo.

```
...
// criação da "pista"  por parte do jogador
// mago_player1 (com id_player1) sendo a sua equipa id_equipa1
name = "pista player";
type = "texto";
property = "info";
utility = ;
msg_info = "sigam directamente para a gruta"

data = new ObjData( name, type, property, utility, msg_info );
place = {posição, local ao qual fica associado a "pista"};

// cria lista com dados para colocar no espaço da equipa
place = {localização actual};
list = new GameObjData( data, place );

// actualização do "objecto" no SI
sinfo.si_updateCGroup( id_jogo1, id_master, list; out: ref_data );
ObjectoD = new TupleDataType( list + ref_data );

// actualiza os dados no espaço partilhado do grupo
v = mago_player1.update( id_player1, id_equipa1, PUBLIC, NO_NOTIFY_UPD, list );

// actualiza info referente ao local/posição
// com "colocação" da pista no local/posição
...
```

Listagem 6.13: Pseudo código da colocação de pistas por parte dos jogadores

Os membros de uma equipa podem enviar mensagens para a equipa adversária (grupo), por exemplo, para reportar algum problema ou somente lançar algum tipo de comentário.

```
...
// envia mensagem à equipa adversária
// emissor player1 , receptor equipa2
v =  mago_player1.send(id_player1, id_equipa2, SPREAD, "acho que estão a perder :P" );
...
```

Listagem 6.14: Pseudo código do envio de membro de uma equipa para a equipa adversária

Ao enviar uma mensagem com o parâmetro SPREAD, é desencadeado, no grupo receptor, um evento para divulgação da mensagem para todos os membros do grupo (sendo este um dos eventos de sistema). Esta mesma sequência pode ser utilizada para enviar mensagens por parte dos membros para o seu próprio grupo.

Uma forma mais simples de jogo/interacção dentro do espaço, pode ser conseguida obrigando os participantes a tirarem fotografias nos diversos locais, tornando-as disponíveis para o grupo, sendo estas passíveis de votação.

Ao nível da partilha de conteúdos (fotos e vídeos) é interessante a utilização de grupos implícitos, podendo estes ser definidos, por exemplo, com base nos participantes que tiraram fotos num dado local. Neste caso, o grupo implícito é gerado com base na característica/propriedade posição, que pode ser explicitamente indicada pelos visitantes ou pode ser obtida automaticamente através do sistema de localização.

6.6 Conclusão

Ao longo deste capítulo foi discutida uma validação da funcionalidade do modelo proposto, tendo por base a sua utilização em diferentes cenários reais de aplicação.

Foi discutida e sugerida a metodologia a ser seguida no desenvolvimento e organização de aplicações baseadas no modelo. Foram também ilustrados alguns dos possíveis padrões de utilização, tendo por base um protótipo de aplicação para o qual foi definido também um suporte de sistema de informação, assim como uma possível arquitectura para a implementação de aplicações.

Tendo por base a integração do modelo nas diversas aplicações analisadas, observou-se que a utilização deste permite a definição de padrões de interacção e de organização de múltiplos utilizadores de um modo simples.

7

Conclusões e trabalho futuro

Conteúdo

Neste capítulo é feito um resumo dos aspectos principais abordados e desenvolvidos ao longo deste trabalho, sendo sintetizadas as principais contribuições. A partir do trabalho desenvolvido, são identificadas linhas de investigação futuras.

7.1 Considerações finais e conclusões

Este trabalho teve como principal motivação a proposta de novas soluções de modelação de aplicações que suportam a interacção entre utilizadores, tendo por base um modelo de programação baseado em grupos de entidades computacionais.

O trabalho aqui apresentado contemplou diversos níveis de desenvolvimento, conforme descrito de seguida:

- identificação de cenários de aplicação;
- análise de soluções para modelação da interacção entre grupos de utilizadores;
- definição de um modelo que tornasse disponíveis diferentes padrões de comunicação e organização e que capturasse a dinâmica de interacção entre grupos de utilizadores;
- implementação de uma arquitectura de suporte ao modelo proposto;
- utilização do modelo proposto na concepção de aplicações e serviços;
- estudo da integração do modelo num conjunto de cenários reais de utilização.

De seguida são apresentadas em mais detalhe, as várias vertentes do trabalho desenvolvido ao longo desta dissertação e as principais contribuições.

Numa primeira fase do trabalho, foi efectuado um levantamento das diversas aplicações e serviços que estão normalmente associados à interacção entre múltiplos utilizadores, tendo como objectivo o estabelecimento de meios de suporte à colaboração e à partilha de informação. Com base nas aplicações e serviços analisados foram identificados os requisitos por forma a caracterizar os padrões de comunicação, de partilha de informação e de organização, que permitam escolher os modelos de comunicação que melhor se adaptem às funcionalidades pretendidas.

De seguida, foi efectuado um estudo de diferentes mecanismos de comunicação e organização, tendo como objectivo avaliar as funcionalidade e características por eles apresentadas. Neste estudo, foi dada uma ênfase especial aos modelos de grupos, pelo facto destes possibilitarem a interacção e coordenação de múltiplas entidades, considerando a natureza dinâmica e organizacional dos cenários de aplicação analisados. Neste estudo foram também contemplados os modelos de comunicação através de espaços partilhados, como mecanismo de suporte à comunicação anónima indirecta entre entidades, de modo a estabelecer interacções num contexto de sistemas distribuídos assíncronos. Também como forma de comunicação assíncrona entre múltiplas entidades e de difusão de informação para um conjunto de utilizadores pré-definido,

foram analisados mecanismos de comunicação baseados em notificação de eventos. A escolha de um modelo de estruturação em grupos, foi motivada pelas potencialidades deste modelo ao nível da capacidade de abstracção, na forma como lida com a dinâmica das interacções e estruturação de múltiplas entidades. Ao associar o conceito de grupo a um espaço de interacção caracterizado por entidades que partilham interesses comuns, contribui-se para uma mais fácil compreensão da dinâmica de aplicações complexas, e facilita-se o seu desenvolvimento, de forma estruturada, modular e incremental.

A abordagem de associar a cada grupo, entendido como uma unidade de estruturação de aplicações, um espaço de partilha apenas acessível aos membros do grupo, já anteriormente analisada em trabalhos como o GroupLog, revelou-se um conceito extremamente interessante. Esta associação é compatível com a noção do grupo como um espaço confinado de interacção onde, para além da difusão, encontramos agora também o acesso ao espaço partilhado, comum aos membros do grupo. A exploração deste conceito, ao nível das aplicações como as identificadas no capítulo 2, é uma das contribuições do trabalho aqui apresentado.

Da análise dos modelos de comunicação e tendo por base os trabalhos de investigação realizados em anteriores propostas de investigação em trabalhos como o GroupLog e o JGroupSpace, surgiu a proposta do modelo MAGO. Esta proposta de modelo, assenta num paradigma de estruturação baseado em grupos integrando de forma unificada, múltiplos mecanismos de comunicação por forma a oferecer uma maior flexibilidade e expressividade. No contexto de um modelo de computação baseado na abstracção de sincronia virtual, assegura-se a consistência das visões dos membros de um grupo, face aos eventos de: modificação da constituição do grupo, comunicação por difusão de mensagens ou eventos, e acesso ao espaço partilhado.

Este modelo suporta um conjunto de primitivas que permitem descrever, de forma simples, os possíveis padrões de utilização na concepção de aplicações interactivas distribuídas, em particular na definição e criação de funcionalidades associadas às estruturas de grupos. Pretendeu-se com este modelo e com base nas primitivas por ele propostas, dar uma maior expressividade ao conceito de grupo, enquanto elemento de organização e estruturação de utilizadores que partilham entre si interesses comuns.

O conceito de grupo implícito surgiu da análise da forma como os utilizadores se tendem a organizar e surge como uma proposta complementar aos serviços oferecidos aos denominados grupos explícitos. Os grupos implícitos distinguem-se pela forma como é estabelecida a sua constituição, não partindo de uma acção explícita de filiação por parte da entidade mas sim de uma análise do seu conjunto de características/atributos. No modelo proposto, este tipo de grupo é gerido de forma a simplificar a sua definição e utilização por parte das aplicações, tendo disponíveis o mesmo tipo de primitivas e de funcionalidades que as oferecidas aos grupos denominados de explícitos,

ou seja aqueles cuja constituição não depende das características/atributos dos seus membros, mas sim de uma acção explicitamente desencadeada por estes para a sua filiação.

As primitivas disponibilizadas pelo modelo podem ser classificadas, de acordo com a suas funcionalidades, em quatro classes distintas: (1) registo e gestão de entidades e grupos; (2) comunicação directa entre entidades; (3) comunicação através de mecanismos de eventos; (4) manipulação do espaço partilhado de tuplos.

Tendo por base o modelo proposto, foi desenvolvida uma arquitectura de suporte e implementada sobre a plataforma JGroupSpace. Sobre esta plataforma, foi implementado um conjunto de primitivas com uma semântica de acordo com a proposta do modelo, assim como um conjunto de componentes que permitem a concepção e realização de aplicações. Um desses componentes é a interface de funcionalidades que dá suporte à interacção com um sistema de informação, assim como os dados e estruturas que dão suporte aos mesmos. A descrição desta interface reveste-se de especial importância devido, principalmente, à sua interligação directa com a forma como é definida a gestão dos grupos implícitos pelo modelo.

Finalmente, foi efectuada uma validação do modelo proposto ao nível da capacidade de representação, tendo por base a sua utilização em diferentes cenários de aplicação. É também apresentada uma aplicação protótipo, como forma de validar a metodologia de desenvolvimento e organização de aplicações baseadas no modelo, assim como a sua integração com soluções tecnológicas já existentes. São ilustrados também possíveis padrões de utilização das funcionalidades e representatividade do modelo, como forma de estruturar e definir novos serviços em aplicações já existentes. Observou-se que a utilização deste modelo permite a definição de padrões de interacção e de organização de aplicações, que lidam com múltiplos utilizadores, de um modo simples e transparente para o programador.

Em resumo, uma das contribuições deste trabalho situa-se na identificação do conjunto de necessidades ao nível dos serviços disponibilizados a grupos de utilizadores, que são muitas vezes mais complexos devido à necessidade de utilização de diferentes sistemas [CAC+05, MS05, CAM+05] [MCC07].

O modelo aqui proposto permite obter, de uma forma integrada os serviços necessários a um sistema de suporte a grupos de utilizadores, que pode ser enquadrado num universo de dispositivos e de formas de utilização abrangentes.

A completa validação desta proposta só será tornada possível no entanto, quando da implementação do sistema num conjunto heterogéneo de aplicações.

7.2 Trabalho futuro

Como já foi referido, o trabalho mais imediato está centrado ao nível das aplicações e na forma como o modelo se integra num conjunto de cenários distintos. Embora já tenha sido analisada a utilização deste modelo em cenários reais de aplicações, no contexto de apoio ao ensino e no contexto de aplicações de apoio ao turismo (InStory), espera-se que novos desenvolvimentos nestas áreas venham a ser prosseguidos.

As características e funcionalidades apresentadas pelo modelo proposto, levam-nos a procurar novas soluções e desafios que promovam de uma forma mais eficiente características tais como a mobilidade, a dependência da localização e a utilização de dispositivos móveis heterogéneos.

Avaliar a forma como os utilizadores interagem de uma forma integrada com os conceitos de grupo implícito e explícito de modo a optimizar os processos de gestão destes grupos, é também um objectivo de trabalho futuro.

Uma vez que o objectivo prioritário na realização do protótipo foi a avaliação da funcionalidade e dos conceito e mecanismos incorporados no modelo, não foram contemplados os estudos das opções de implementação que melhorassem a eficiência ao nível das comunicações e possíveis optimizações ao nível da execução das primitivas do modelo, aspectos que serão considerados em trabalho futuro.

Estão também previstos desenvolvimentos na definição e gestão dos grupos implícitos, em particular na forma como são especificadas as regras para a sua formação, de modo a permitir regras de formação mais complexas e que permitam uma maior expressividade, já que a actual implementação tem por base um sistema simples de verificação de condições.

A

Componentes da arquitectura do sistema

Conteúdo

A.1 Introdução

Neste anexo são ser descritos os componentes de software que constituem a base da arquitectura que implementa o modelo MAGO: a entidade e o representante de grupo. No anexo seguinte são descritos os servidores do sistema.

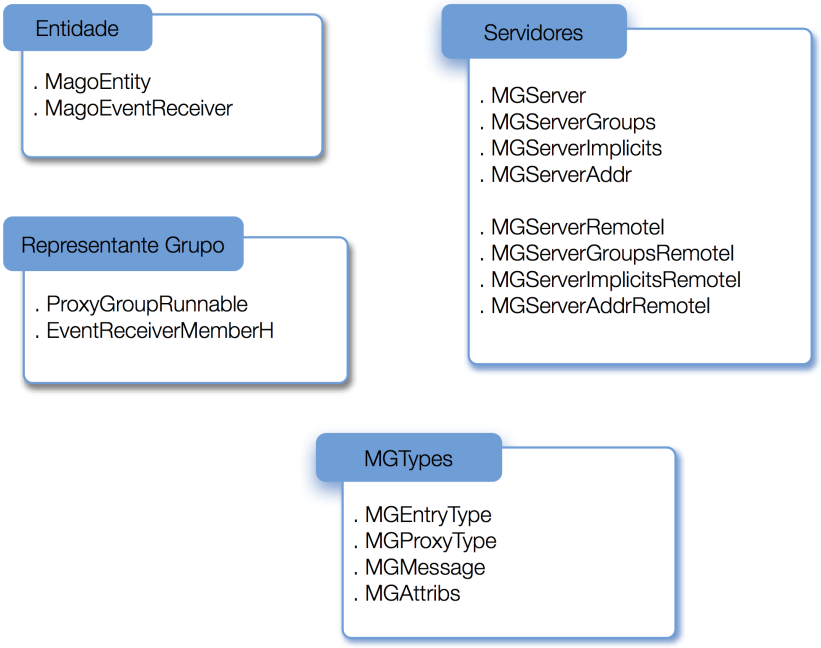

Figura A.1: Elementos base do sistema

A.2 Entidades

- *MagoEntity* - esta classe implementa o conjunto de primitivas/métodos disponibilizados pelo modelo MAGO. Gere as estruturas que suportam a informação relativa às entidades e à sua filiação nos diversos grupos.

- *MagoEventReceiverH* - definição dos handlers para lidar com os eventos de sistema e de aplicação por parte de cada entidade.

- *EntityGroupType* - classe com informação referente à entidade, enquanto membro de um grupo.

A.2.1 Classe MagoEntity

Esta classe implementa a interface onde são disponibilizadas as primitivas do modelo MAGO e representa uma entidade elementar no sistema.

```
public class MagoEntity { ... }
```

Campos

- *EntityName* - nome que identifica a entidade no sistema

```
public String EntityName;
```

- *myUAddress* - Endereço lógico atribuído à entidade no grupo universal

```
public MGAddress myUAddress
```

- *numberEntityGroups* - número de grupos em que a entidade se encontra filiada

```
public int numberEntityGroups;
```

- *myGroupsTable* - tabela com a informação relativa à filiação da entidade nos diversos grupos

```
// tabela com tipo onde está info sobre os grupos a que pertence
// – group name
// – proxy group address
// – group space of the group
// – entity address on the group
// – entity status on that group
static HashMap<String,EntityGroupType> myGroupsTable =
                                              new HashMap<String,EntityGroupType>();
```

- *myEventsTable* - tabela com a informação relativa aos diferentes tipos de eventos anunciados e subscritos pela entidade nos diversos grupos

```
// tabela com tipo onde está info sobre os tipos de eventos suportados
// – group name
// – event ID
// – event state
// – event info
// – event method
static HashMap<String,String,String,String,String> myEventsTable =
                                new HashMap<String,String,String,String,String>();
```

Métodos

- *register*

```
public int register (String newEntityName, AppDefType entity)
```

– **Descrição**:
Efectua o registo da entidade no sistema, ou seja torna a entidade membro do grupo universal, definindo um espaço e endereço lógico absoluto nesse grupo.

– **Parâmetros**:

∗ *newEntityName* - nome próprio definido para a entidade;

∗ *entity* - objecto com informação, definida pela aplicação para caracterizar a entidade no sistema.

– **Retorna**:
Valor positivo com o identificador se a operação de registo da entidade for concluída com sucesso.

- *unregister*

```
public int unregister (String uniqueName)
```

– **Descrição**:
Efectua a remoção da entidade do sistema, ou seja do grupo universal.

– **Parâmetros**:

∗ *uniqueName* - nome que identifica a entidade no sistema.

– **Retorna**:
Valor positivo se a operação de saída da entidade for concluída com sucesso.

- *create*

```
public int create (String entityID, String group_name, int gtype, int filiation, int
     maxm, MGProxyType gt, MGAttribs, String groupID)
```

– **Descrição**:
Efectua a criação de um novo grupo no sistema de acordo com os argumentos que são passados, lançando um representante e atribuindo-lhe um espaço e um endereço lógico absoluto a partir do qual o grupo passa a poder ser contactado enquanto entidade composta do sistema.

– **Parâmetros**:

∗ *entityID* - identificador da entidade criadora do novo grupo;

∗ *group_name* - nome lógico atribuído ao grupo;

∗ *gtype* - tipo de grupo a ser criado (EXPLICT ou IMPLICIT);

∗ *filiation* - tipo de política de filiação a ser atribuída ao novo grupo (FREE, BY_ONE, MAJORITY, BY_LIST, BY_CREATOR);

  - *maxm* - número máximo de membros que podem ser admitidos no grupo;
  - *gt* - estrutura com informação própria associada ao grupo;
  - *groupID* - estrutura com conjunto de atributos/valores que caracterizam um grupo implícito;
- **Retorna**:
  Valor positivo se a operação de criação de novo grupo for concluída com sucesso.

- *destroy*

```
public int destroy (String uniqueName, String nome_grupo, int aviso, String msg)
```

- **Descrição**:
  Efectua a remoção de um grupo do sistema de acordo com o tipo de notificação especificado no argumento *aviso*.
- **Parâmetros**:
  - *uniqueName* - identificador do grupo a remover;
  - *nome_grupo* - nome lógico atribuído ao grupo;
  - *aviso* - tipo de encerramento de grupo a ser executado (NO_NOTIFICATION, NOTIFICATION, NOTIFICATION_ACK);
  - *msg* - no caso de encerramento do tipo NOTIFICATION, mensagem a ser enviada aos restantes membros do grupo, quando do encerramento deste.
- **Retorna**:
  Valor positivo, se a operação de remoção de grupo for concluída com sucesso.

- *join*

```
public int join (String entidadeID, String nome_grupo, String info, MGAddress gAddr)
```

- **Descrição**:
  Efectua a filiação de uma entidade num grupo de acordo com a política de filiação definida para o grupo quando da sua criação.
- **Parâmetros**:
  - *entidadeID* - identificador da entidade;
  - *nome_grupo* - nome lógico atribuído ao grupo, a partir do qual obtém o seu identificador (endereço lógico absoluto);

  - *info* - informação opcional que pode ser enviada quando da filiação num grupo, por exemplo no caso do grupo ter definida uma política de filiação do tipo BY_LIST este campo é utilizado para passar a palavra de acesso;
  - *gAddr* - endereço lógico local atribuída à entidade no grupo em que se filiou.

- **Retorna**:
  Valor positivo se a operação de filiação no grupo for concluída com sucesso.

- *leave*

```
public int leave (String entidadeID, String nome_grupo)
```

- **Descrição**:
  Efectua a saída de uma entidade de um grupo, ou seja a entidade deixa de ser membro do grupo indicado. Operação somente válida para grupos explícitos.
- **Parâmetros**:
  - *entidadeID* - identificador da entidade;
  - *nome_grupo* - nome lógico atribuído ao grupo, a partir do qual obtém o seu identificador (endereço lógico absoluto).
- **Retorna**:
  Valor positivo se a operação de saída de entidade do grupo for concluída com sucesso.

- *membership*

```
public Vector membership ( String nome_grupo )
```

- **Descrição**:
  Efectua consulta de membros filiados no grupo indicado.
- **Parâmetros**:
  - *nome_grupo* - nome lógico atribuído ao grupo, a partir do qual obtém o seu identificador (endereço lógico absoluto).
- **Retorna**:
  Vector com identificadores de entidades filiadas no grupo e respectivo estados de actividade.

- *set_mode*

```
public int set_mode ( String entidadeID, String nome_grupo, String modo )
```

  - **Descrição**:
    Actualiza modo do membro filiado no grupo indicado.
  - **Parâmetros**:
    - *entidadeID* - identificador da entidade;
    - *nome_grupo* - nome lógico atribuído ao grupo, a partir do qual obtém o seu identificador (endereço lógico absoluto);
    - *modo* - modo de actividade a atribuir à entidade no grupo indicado (ONLINE ou OFFLINE).
  - **Retorna**:
    Valor positivo, se a operação de modificação de estado da entidade no grupo for concluída com sucesso.

- *send*

```
public int send( String entidadeID, String entidadeRcv, String sendT, String metodo,
    MGMessage msg )
```

  - **Descrição**:
    Envia uma mensagem para um destinatário cujo identificador é indicado em *entidadeRcv*.
  - **Parâmetros**:
    - *entidadeID* - identificador da entidade emissora;
    - *entidadeRcv* - identificador da entidade receptora da mensagem;
    - *sendT* - tipo de *send*, neste caso é do tipo M, nome do método passado como argumento da função;
    - *metodo* - nome do método de recepção a ser invocado no receptor;
    - *msg* - mensagem a ser enviada
  - **Retorna**:
    Valor positivo, se a operação de modificação de estado da entidade no grupo for concluída com sucesso.

- *send*

```
public int send( String entidadeID, String entidadeRcv, String sendT, int notifyT,
    MGMessage msg )
```

  - **Descrição**:
    Envia uma mensagem para uma entidade ou grupo identificada por *entidadeRcv*.
  - **Parâmetros**:
    - *entidadeID* - identificador da entidade emissora;
    - *entidadeRcv* - identificador da entidade receptora da mensagem (entidade ou grupo);
    - *sendT* - tipo de *send*, neste caso é do tipo S método predefinido pela aplicação, sendo a leitura efectuada por *receive*;
    - *notifyT* - tipo de notificação a ser desencadeada no caso de entidade receptora ser um grupo (PASSIVE, SPREAD e NOTIFY);
    - *msg* - mensagem a ser enviada.
  - **Retorna**:
    Valor positivo, se a operação de envio de mensagem for concluída com sucesso.

- *receive*

```
public int receive( int rcvT, MGMessage msg )
```

  - **Descrição**:
    Efectua a leitura de uma mensagem dirigida a uma entidade elementar.
  - **Parâmetros**:
    - *rcvT* - tipo de leitura pode ser bloqueante ou não bloqueante (BLOCK ou NO_BLOCK);
    - *msg* - mensagem lida.
  - **Retorna**:
    Valor positivo, se a operação de recepção de mensagem for concluída com sucesso.

- *receive*

```
public int receive( int rcvT, String grupoID, String msgID, String filtro, MGMessage
    msg )
```

  - **Descrição**:
    Efectua a leitura de uma mensagem dirigida a uma entidade elementar.

– **Parâmetros**:

* *rcvT* - tipo de leitura pode ser bloqueante ou não bloqueante (BLOCK ou NO_BLOCK);
* *grupoID* - identificador do grupo;
* *msgID* - identificador de mensagem, no caso de ser null, lê qualquer mensagem;
* *filtro* - filtro definido, que permite efectuar uma leitura selectiva da fila de mensagens do grupo; no caso de ser null: filtro desactivado;
* *msg* - mensagem lida.

– **Retorna**:
Valor positivo, se a operação de recepção de mensagem for concluída com sucesso.

- *advertise*

```
public int advertise( String entidadeID, String grupoID, int advT, String eventoID,
    String info )
```

– **Descrição**:
Efectua o anúncio de um novo tipo de evento no grupo indicado, por parte da entidade, sendo passado por argumento o identificador do novo evento.

– **Parâmetros**:

* *entidadeID* - identificador da entidade;
* *grupoID* - identificador do grupo;
* *advT* - tipo de anúncio de evento que deve ser desencadeado (PASSIVE e ACTIVE);
* *eventoID* - identificador do evento;
* *info* - informação adicional referente ao novo tipo de evento.

– **Retorna**:
Valor positivo, se a operação de anúncio de novo tipo de evento tiver sido bem sucedida.

- *unadvertise*

```
public int unadvertise( String entidadeID, String grupoID, String eventoID )
```

– **Descrição**:
Desactiva tipo de evento previamente anunciado no grupo pela entidade.

– **Parâmetros**:

* *entidadeID* - identificador da entidade;
* *grupoID* - identificador do grupo;
* *eventoID* - identificador do evento.

– **Retorna**:
Valor positivo, se a operação de desactivação de tipo de evento tiver sido bem sucedida.

- *check_adv*

```
public Vector check_adv( String entidadeID, String grupoID )
```

– **Descrição**:
Obtém os tipos de evento anunciados no grupo pelas diversas entidades membros.

– **Parâmetros**:

* *entidadeID* - identificador da entidade que efectuou os pedidos no Grupo Universal.
* *grupoID* - identificador do grupo;

– **Retorna**:
Vector com informação referente aos tipos de eventos anunciados no grupo indicado, com o tipo de evento, entidade que o anunciou e método de atendimento sugerido.

- *subscribe*

```
public int subscribe( String grupoID, String eventoID, String metodo )
```

– **Descrição**:
Entidade torna-se subscritora de um tipo de evento previamente anunciado no grupo, indicando qual o método de atendimento que a entidade desencadeia quando da ocorrência do evento, actualizando a informação na estrutura de eventos do grupo e da entidade.

– **Parâmetros**:

* *grupoID* - identificador do grupo;
* *eventoID* - identificador do evento;
* *metodo* - nome do método de atendimento a ser desencadeado, pela entidade, quando da ocorrência deste tipo de evento.

– **Retorna**:
Valor positivo, se a operação de subscrição do tipo de evento tiver sido bem sucedida.

- *unsubscribe*

```
public int unsubscribe( String grupoID, String entidadeID, String eventoID )
```

– **Descrição**:
Entidade deixa de ser subscritora de um tipo de evento previamente anunciado no grupo, actualizando a informação na estrutura de eventos do grupo e da entidade.

– **Parâmetros**:

∗ *grupoID* - identificador do grupo;

∗ *entidadeID* - identificador da entidade;

∗ *eventoID* - identificador do evento;

– **Retorna**:
Valor positivo, se a operação de desactivação de subscrição do tipo de evento por parte da entidade tiver sido bem sucedida.

- *publish*

```
public int publish( String grupoID, MGEventType evento )
```

– **Descrição**:
Entidade publica um evento, ou seja desencadeia o processo de difusão do evento pelos membros do grupo.

– **Parâmetros**:

∗ *grupoID* - identificador do grupo;

∗ *evento* - objecto evento, com informação referente ao tipo de evento, ao seu produtor e informação adicional associada ao evento.

– **Retorna**:
Valor positivo, se a operação de publicação de evento por parte da entidade tiver sido bem sucedida.

- *update*

```
public int update( String entidadeID, String grupoID, int acessoT, int notifT,
    ObjectData dados )
```

– **Descrição**:
Entidade coloca dados no espaço partilhado do grupo, estabelecendo se os dados são públicos ou privados e de que forma os restantes membros do grupo serão notificados da disponibilização de dados para consulta.

– **Parâmetros**:

* *entidadeID* - identificador da entidade produtora dos dados;
* *grupoID* - identificador do grupo onde vai ser disponibilizada a informação;
* *acessoT* - caracterização do tipo de dados, se PRIVATE ou PUBLIC;
* *notifT* - tipo de notificação que deve ser desencadeada ao membros do grupo (NOTIFY_UPD ou NO_NOTIFY_UPD);
* *dados* - objecto com os dados definidos pela aplicação a serem colocados no espaço partilhado.

– **Retorna**:
Valor positivo, com identificação atribuída aos dados se a operação de escrita destes no espaço partilhado tiver sido bem sucedida.

- *set_access*

```
public int set_access( String entidadeID, String grupoID, int acessoT, String dataID
    )
```

– **Descrição**:
Entidade altera o tipo de acesso definido para os dados escritos no espaço, somente se entidade for a produtora dos dados.

– **Parâmetros**:

* *entidadeID* - identificador da entidade produtora dos dados;
* *grupoID* - identificador do grupo onde vai ser disponibilizada a informação;
* *acessoT* - caracterização do tipo de dados, se PRIVATE ou PUBLIC;
* *dataID* - identificador atribuído aos dados quando da sua escrita.

– **Retorna**:
Valor positivo, se a alteração do tipo de acesso aos dados tiver sido efectuada com sucesso.

- *find*

```
public int find( String entidadeID, String grupoID, String produtorID, int sinc,
    MGAttribs pesquisa )
```

- **Descrição**:
  Efectua a pesquisa do espaço de acordo com os critérios de pesquisa definidos em *pesquisa*. Este método actualiza a estrutura interna com todo os tuplos do espaço que satisfizeram o critério de pesquisa especificado.
- **Parâmetros**:
  - *entidadeID* - identificador da entidade;
  - *grupoID* - identificador do grupo onde vai ser efectuada a pesquisa;
  - *produtorID* - identificador da entidade produtora dos dados, ou seja que efectuou a escrita;
  - *sinc* - indicação se a consulta é bloqueante ou não (BLOCK ou NO_BLOCK);
  - *pesquisa* - estrutura com os atributos/valores que definem o critério de pesquisa a ser desencadeado e que permitem a construção do tuplo de pesquisa.
- **Retorna**:
  Devolve o número de tuplos do espaço que satisfizeram os critérios de pesquisa; no caso da opção de BLOCK, só retorna quando existir pelo menos um tuplo que obedeça às condições.

- *consult*

```
public int consult( int num, ObjectData dados )
```

  - **Descrição**:
    Efectua leitura dos dados da estrutura interna que foi actualizada após uma operação de $find$.
  - **Parâmetros**:
    - *num* - identificador dos dados na estrutura interna que foi previamente actualizada por uma operação de $find$. O valor de *num* encontra-se entre 0 e o valor retornado pela operação $find$;
    - *dados* - dados lidos do espaço;
  - **Retorna**:
    Número de dados lidos do espaço que satisfazem os critérios de pesquisa indicados.

- *get*

```
public int get( int num, ObjectData dados )
```

– **Descrição**:
Efectua remoção dos dados da estrutura interna que foi actualizada após uma operação de *find*.

– **Parâmetros**:

∗ *num* - identificador dos dados na estrutura interna que foi previamente actualizada por uma operação de *find*. O valor de *num* encontra-se entre 0 e o valor retornado pela operação *find*;

∗ *dados* - dados removidos da estrutura interna.

– **Retorna**:
Número de dados removidos da estrutura interna que se encontravam na posição indicada da estrutura.

- *consult*

```
public int consult( String entidadeID, String grupoID, String produtorID, int sinc,
    MGAttribs pesquisa, ObjectData dados)
```

– **Descrição**:
Efectua leitura de dados do espaço partilhado de acordo com os critérios de pesquisa indicados; se definida a opção de NO_BLOCK e não existirem dados que satisfaçam os critérios de pesquisa retorna de imediato com objecto de dados a null.

– **Parâmetros**:

∗ *entidadeID* - identificador da entidade que está a efectuar a pesquisa;

∗ *grupoID* - identificador do grupo onde vai ser efectuada a pesquisa;

∗ *produtorID* - identificador da entidade produtora dos dados, ou seja que efectuou a escrita;

∗ *sinc* - indicação se a consulta é bloqueante ou não (BLOCK ou NO_BLOCK);

∗ *pesquisa* - estrutura com os atributos/valores que definem o critério de consulta a ser desencadeado;

∗ *dados* - dados lidos do espaço de acordo com formato especificado pela aplicação.

– **Retorna**:
Dados lidos do espaço que satisfazem os critérios de pesquisa indicados.

- *get*

```
public int get( String entidadeID, String grupoID, String produtorID, int sinc,
    MGAttribs pesquisa, ObjectData dados)
```

– **Descrição**:
Efectua leitura e remoção de dados do espaço partilhado de acordo com os critérios de pesquisa indicados; se definida a opção de NO_BLOCK e não existirem dados que satisfaçam os critérios retorna de imediato com objecto de dados a null.

– **Parâmetros**:
* *entidadeID* - identificador da entidade que está a efectuar a pesquisa;
* *grupoID* - identificador do grupo onde vai ser efectuada a pesquisa;
* *produtorID* - identificador da entidade produtora dos dados, ou seja que efectuou a escrita;
* *sinc* - indicação se a consulta é bloqueante ou não (BLOCK ou NO_BLOCK);
* *pesquisa* - estrutura com os atributos/valores que definem o critério de consulta a ser desencadeado;
* *dados* - dados lidos do espaço de acordo com formato especificado pela aplicação.

– **Retorna**:
Dados lidos do espaço que satisfazem os critérios de pesquisa indicados.

- *listMyGroups*
Método auxiliar que efectua a leitura/escrita dos grupos a que a entidade pertence actualmente

```
public int listMyGroups()
```

A.2.2 MagoEventReceiveH

Esta classe lida com o tratamento de eventos por parte da entidade, implementando os *handlers* que são activados pelos diferentes tipos de eventos em cada grupo.

```
public class MagoEventReceiveH implements JGEventHandler {...}
```

Métodos

- *handleEvent*

```
public void handleEvent(Object obj)
```

– **Descrição**:
Handler de suporte aos vários eventos da entidade

– **Parâmetros**:

∗ *obj* - informação associada ao evento

A.3 Representante de grupo

O representante de grupo é definido ao nível de sistema por duas classes: *ProxyGroupRunnable* e *EventReceiverMemberH*.

A.3.1 ProxyGroupRunnable

A classe que representa este elemento (*ProxyGroupRunnable*) implementa *Runnable* e é lançada pelo servidor de grupos e possui a seguinte declaração:

```
public class ProxyGroupRunnable implements Runnable {...}
```

Campos

Os campos definidos contêm o conjunto de informação que caracteriza o estado actual do grupo, tais como, o conjunto de membros que já se filiaram no grupo e o estado de actividade actualmente observado por estes.

- Informação referente à caracterização do grupo enquanto elemento do grupo universal

```
// Group Global information
public String groupName;              // nome do grupo
public int groupType;                 // tipo de grupo, EXPLICIT ou IMPLICIT
public JGroupSpace jgs;               // espaço criado para interacções globais
public JGSAddress proxyUAddress;      // endereço físico das interacções globais
public MGAddress myGlobalAddress;     // endereço lógico das interacções globais
```

- Informação local do grupo

```
// Group Local information
public JGroupSpace mygroup;           // espaço criado para interacções locais
public JGSAddress proxyGroupAddress;  // endereço físico das interacções locais
public MGAddress  myLocalAddress;     // endereço lógico das interacções locais
```

- Informação geral do grupo referente à sua constituição

```
    // Numero de membros actuais filiados no grupo
    public int nMembers;
    // Informação referente aos membros do grupo
    public HashMap<String,MGEntryType> groupMembersTable =
```

```
                        new HashMap<String,MGEntryType>();

    // Informação referente aos eventos definidos para o grupo
    static HashMap<String,String,String,String,String> groupEventsTable =
                        new HashMap<String,String,String,String,String>();

    // Informação de configuração do grupo
    private String cfg;

    // Tipo de filiação configurado para o grupo
    private int joinPolicy;

    // Endereço lógico da entidade criadora do grupo
    private MGAddress creatorAddr;

    // Informação de configuração do grupo
    private MGProxyType newProxyGroup;
```

Métodos

- *joinProtocol*

```
public int joinProtocol ( String entidadeID, String info)
```

  - **Descrição**:
    Executa o processo de admissão ao grupo de acordo com a política de filiação definida para o grupo.
  - **Parâmetros**:
    - *entidadeID* - identificador da entidade;
    - *info* - informação fornecida pela entidade, quando efectua o pedido de filiação ao grupo.
  - **Retorna**:
    Valor positivo se a filiação tiver sido efectuada com sucesso.

- *run*

```
public void run()
```

  - **Descrição**:
    Fluxo de execução do representante de grupo: numa primeira fase é efectuado o conjunto de configurações definido para o tipo de grupo; numa segunda, fica a aguardar mensagens.

A.3.2 EventReceiverMemberH

Classe que implementa *JGEventHandler*, responsável por desencadear o atendimento dos diversos tipos de eventos configurados e definidos para o grupo.

```
public class EventReceiveMemberH implements JGEventHandler {...}
```

Campos

Com informação referente ao grupo a que está associado

```
public String groupName;
public ProxyGroupRunnable myProxy;
public MGAddress myGlobalAddress;
public MGAddress  myLocalAddress;
```

Métodos

- *handleEvent*

  ```
  public void handleEvent(Object obj)
  ```

  - **Descrição**:
    Handler de suporte aos vários eventos suportados pelo grupo
  - **Parâmetros**:
    - *obj* - informação associada ao evento

A.4 Tipos de dados base do sistema

Tipos de base definidos no sistema *Mago*:

- *MagoEntryType* - classe com informação que caracteriza uma entidade no sistema ao nível dos servidores;
- *MagoProxyType* - classe com informação que caracteriza um grupo no sistema ao nível dos servidores;
- *MagoMessageType* - tipo para lidar com mensagens e que obedece ao formato especificado;
- *MagoAttribs* - tipo para lidar com atributos e que obedece ao formato especificado (nome,valor).

B

Servidores do sistema

Conteúdo

B.1 Introdução

Este anexo contém uma breve descrição dos quatro serviços de base do sistema:

- servidor grupo universal (**MGServer**);
- servidor de grupos (**MGServerGroups**);
- servidor de grupos implícitos (**MGServerImplicits**);
- servidor de endereços (**MGServerAddr**).

Para todos os serviços tomou-se a opção de os implementar através de um servidor, ou seja a opção de centralizar esses elementos.

Todos os servidores são acedidos através de RMI - Remote Method Invocation. O esquema de programação com RMIs (figura B.1) envolve o desenvolvimento de uma `interface`, onde são declarados os métodos que vão estar disponíveis remotamente e a implementação, no servidor, dessa interface.

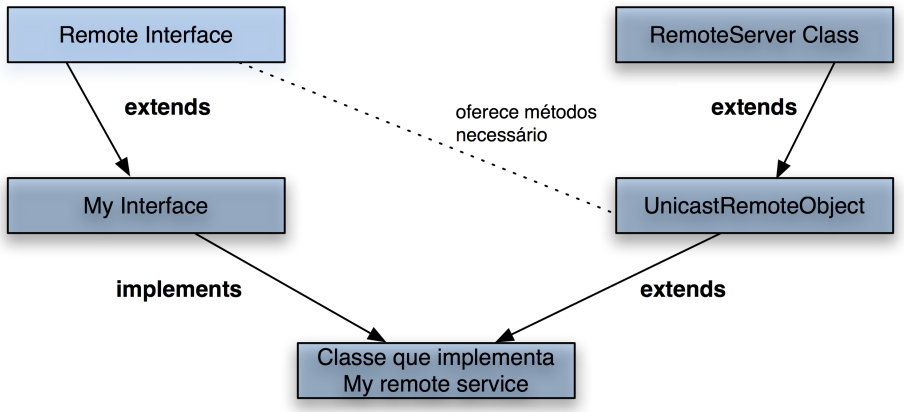

Figura B.1: Esquema de definição da interface remota

```
import java.rmi.Remote;
import java.rmi.RemoteException;
...

public interface MyRemote extends Remote {
        public int ServiceExample ( int x ) throws RemoteException;
        ...
}
```

Listagem B.1: Exemplo de definição de uma interface

Do lado dos clientes, que invocam os serviços remotos, é necessário que estes conheçam o *hostname* ou endereço IP da máquina que executa o servidor e o nome do objecto remoto, disponível através da interface.

B.2 Servidor grupo universal - *MGServer*

Este é o servidor principal de todo o sistema, responsável pela gestão do Grupo Universal, a que todos as entidades do sistema pertencem. Tem a seu cargo as tarefas de gestão das entradas e saídas do sistema (*register*() e *unregister*()) assim como o desencadear do processo de criação e de remoção de grupos (*create*() e *destroy*()). Para a execução de grande parte das primitivas desenvolvidas no modelo existe a necessidade de interagir com este elemento do sistema. Dado que este servidor é o responsável também pela gestão das estruturas com as identificações/endereços das entidades e grupos do sistema, disponibiliza ainda um conjunto de métodos, que podem ser invocados remotamente pela aplicação, e que permitem obter informação referente ao estado actual do sistema no que se refere às entidades e grupos que dele fazem parte assim como da sua identificação/endereços.

Declaração

Esta classe implementa a interface remota *MGServerRemoteI*.

```
public class MGServer extends UnicastRemoteObject implements MGServerRemoteI
```

Campos

- ProxiesTable - Tabela com informação referente aos grupos criados

```
static HashMap<String,MGProxyType> ProxiesTable = new HashMap<String,MGProxyType>();
```

- EntitiesTable - Tabela com informação referente às entidades registadas

```
static HashMap<String,MGEntryType> EntitiesTable = new HashMap<String,MGEntryType>();
```

Métodos

- *getRegistedEntities*

```
public Vector getRegistedEntities () throws RemoteException;
```

  - **Descrição**:
    Obtém informação referente a todas as entidades elementares registadas no sistema; se vector vazio significa que não existem actualmente entidades registadas no sistema.
  - **Retorna**:
    Vector de entidades registadas, elementos do tipo MGEntityType, com informação referente à entidade e ao seu estado de actividade actual.

- *getCreatedGroups*

```
public Vector getCreatedGroups () throws RemoteException;
```

  - **Descrição**:
    Obtém informação referente a todas aos grupos criados no sistema; se vector vazio significa que não existe actualmente nenhum grupo criado.
  - **Retorna**:
    Vector de grupos criados, elementos do tipo MGProxyType, com informação geral referente ao grupo.

- *createGroup*

```
public void createGroup (MGAddress addrCreator, MGProxyType newGroup, String info)
    throws RemoteException;
```

  - **Descrição**:
    Efectua o pedido de criação de novo grupo ao servidor de grupos (*MGServerGroups*) e actualiza dados da estrutura de grupos/entidades
  - **Parâmetros**:
    * addrCreator - endereço da entidade que desencadeou o processo de criação do grupo
    * newGroup - dados referentes ao novo grupo a ser criado
    * info - informação adicional a ser passada ao representante do novo grupo

- *setEntityEntry*

```
public void setEntityEntry(String entityName, MGEntryType entityEntry) throws
    RemoteException;
```

  - **Descrição**:
    Cria entrada de nova entidade na sua estrutura e procede à actualização dos seus dados.
  - **Parâmetros**:
    * entityName - nome (login) da entidade;
    * entityEntry - objecto com os dados referentes à entidade.

- *setGroupEntry*

```
public void setGroupEntry(String groupName, MGProxyType proxyEntry) throws
    RemoteException;
```

– **Descrição**:
Cria entrada do novo grupo na sua estrutura e procede à actualização dos dados.

– **Parâmetros**:

* groupName - nome que identifica o grupo;
* groupEntry - objecto com os dados referentes ao grupo.

B.3 Servidor de grupos - *MGServerGroups*

Este servidor tem a seu cargo a tarefa de gestão de criação de novos grupos. Quando é efectuado um pedido de criação de um novo grupo, é desencadeada uma chamada ao servidor de grupo universal que por sua vez contacta este servidor, que é o responsável pelo lançamento do processo do representante do novo grupo através do método *launchProxy*.

Declaração

Esta classe implementa a interface remota *MGServerGroupsRemoteI*.

```
public class MGServerGroups extends UnicastRemoteObject implements MGServerGroupsRemoteI
    {...}
```

Campos

- número de grupos explícitos já criados

```
int ngroupsE;
```

- número de grupos implícitos já criados;

```
int ngroupsI;
```

Método principal da classe

- *launchProxy*

```
public int launchProxy (MGAddress addrCreator, MGProxyType newGroup, String info)
    throws RemoteException;
```

– **Descrição**:
Lança o processo (*thread*) do representante do novo grupo, de acordo com a configuração especificada.

– **Parâmetros**:
  * addrCreator - endereço da entidade que desencadeou o processo de criação do grupo;
  * newGroup - dados referentes ao novo grupo a ser criado;
  * info - informação adicional a ser passada ao representante do novo grupo.

– **Retorna**:
  Valor de 0 se a operação de lançamento do representante foi concluída com sucesso.

B.4 Servidor de grupos implícitos - *MGServerImplicits*

Este servidor tem como função gerir os pedidos de formação de grupos implícitos. Possui uma interface que lhe permite lidar com os pedidos de novos grupos implícitos, recebendo um conjunto de parâmetros com base nos quais efectua uma consulta ao sistema de informação, com vista à obtenção de uma lista de potenciais participantes, ou seja, a constituição do grupo.

Declaração

Esta classe implementa a interface remota *MGServerImplicitsRemoteI*.

```
public class MGServerImplicits extends UnicastRemoteObject implements
    MGServerImplicitsRemoteI {...}
```

Campos

- ImplicitsTable - Tabela com informação dos grupos implícitos existentes

```
static HashMap<String, MGImplicits> ImplicitsTable = new HashMap<String, MGImplicits>();
```

Métodos

- *findImplicitMembers*

```
public Vector findImplicitMembers (String gid, MGAttribs Attribs) throws
    RemoteException;
```

  – **Descrição**:
    Consulta o SI por forma a encontrar todos os membros do sistema que verificam as condições indicadas em Attribs, com vista a torná-los membros no novo grupo.

  - **Parâmetros**:
    - *gid*: nome do grupo;
    - *Attribs*: lista de atributos a serem observados pelo novo grupo.
  - **Retorna**:
    Vector com identificadores de todos os candidatos a membro do novo grupo

- *updateImplicits*

```
public MGList updateImplicits ( String id, MGAttribs Attribs ) throws RemoteException;
```

  - **Descrição**:
    Dado o novo conjunto de atributos de uma dada entidade, determina os grupos da qual esta deve ser removida e em quais esta deve ser inserida, através de consulta aos dados referentes à entidade e aos grupos implícitos, presentes no sistema de informação.
  - **Parâmetros**:
    - *id*: identificador da entidade;
    - *Attribs*: lista dos novos atributos da entidade.
  - **Retorna**:
    Estrutura com o conjunto dos grupos dos quais a entidade deve ser removida e o conjunto dos grupos nos quais esta deve ser inserida (filiada).

B.5 Servidor de endereços - *MGServerAddr*

A tarefa deste servidor consiste em efectuar um mapeamento entre os endereços atribuídos pela plataforma JGroupSpace e os endereços lógicos absolutos no sistema. Este servidor garante que os endereços lógicos absolutos atribuídos são únicos e que existe uma correspondência directa entre esses endereços e os endereços atribuídos pela plataforma JGroupSpace. Os endereços lógicos são estáticos, ou seja, após a sua atribuição não sofrem modificações.

O servidor de endereços é responsável por:

- gerir a atribuição de endereços lógicos;
- efectuar o mapeamento entre os endereços lógicos absolutos e os endereços físicos (atribuídos pelo JGroupSpace);
- responder a pedidos de informação referentes aos endereços de uma dada entidade elementar ou grupo.

Declaração

Esta classe implementa a interface remota *MGServerAddrRemoteI*.

```
public class MGServerAddr extends UnicastRemoteObject implements MGServerAddrRemoteI
```

Campos

- AddressTable - Tabela com informação referente ao mapeamento dos endereços lógicos e físicos

```
static HashMap<String,MGAddress,JGSAddress> AddressTable = new HashMap<String,
    MGAddress,JGSAddress>();
```

Métodos

- *newMGAddress*

```
public MGAddress newMGAddress (String id, JGSAddress add1) throws RemoteException;
```

  - **Descrição**:
    Atribui um endereço lógico, actualizando o mapeamento entre endereço lógico e físico.
  - **Parâmetros**:
    - *id*: nome identificador do elemento;
    - *add*1: endereço físico atribuído na plataforma JGroupSpace.
  - **Retorna**:
    Novo endereço lógico atribuído (em fase de protótipo esse valor é um valor inteiro).

- *setMGAddress*

```
public int setMGAddress (JGSAddress add1, MGAddress add2) throws RemoteException;
```

  - **Descrição**:
    Actualiza o mapeamento interno do endereços físico e lógicos, ou seja actualiza o novo valor do endereço físico.
  - **Parâmetros**:
    - *addr*1: endereço actual atribuído na plataforma JGroupSpace;
    - *add*2: endereço lógico.
  - **Retorna**:
    Valor inteiro 0 se o mapeamento efectuado com sucesso; -1 se endereço lógico inexistente.

- *getMGAddress*

```
public MGAddress getMGAddress (JGSAddress add1) throws RemoteException;
```

  - **Descrição**:
    Dado um endereço físico, retorna o endereço lógico atribuído.
  - **Parâmetros**:
    - *addr*1: endereço actual atribuído na plataforma JGroupSpace.
  - **Retorna**:
    Endereço lógico referente ao endereço físico passado como argumento.

- *getJGSAddress*

```
public JGSAddress getJGSAddress (MGAddress add1) throws RemoteException;
```

  - **Descrição**:
    Dado um endereço lógico, retorna o endereço físico actualmente atribuído.
  - **Parâmetros**:
    - *addr*1: endereço lógico.
  - **Retorna**:
    Endereço físico actualmente associado ao endereço lógico.

- *getIDAddress*

```
public Vector getIDAddress (String id) throws RemoteException;
```

  - **Descrição**:
    Dado um identificador de uma entidade (ou grupo), retorna vector com todos os endereços lógicos já atribuídos a essa entidade.
  - **Parâmetros**:
    - *id*: identificador da entidade elementar ou grupo.
  - **Retorna**:
    Vector com todos os endereços lógicos atribuídos à entidade, ou null caso não exista nenhum.

C

Suporte da interacção com o sistema de informação

Conteúdo

C.1 Descrição

O esquema de interacção com o sistema de sistema de informação encontra-se representado na figura C.1.

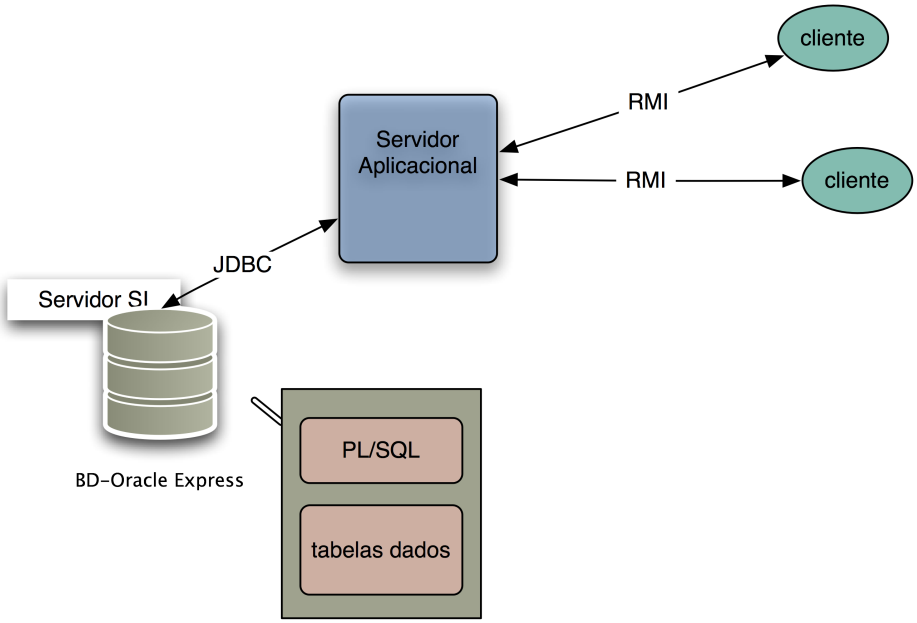

Figura C.1: Esquema de acesso ao sistema de informação

O acesso ao sistema de informação por parte do código Java efectua-se através de uma conexão JDBC. A figura C.1 descreve sumariamente o esquema de acesso onde estão destacados o servidor aplicacional, onde o código Java é executado e o servidor de SI, baseado numa base de dados (SGBD) Oracle Express.

Ao nível do SI é realçado o facto de o acesso aos dados ser mediado por funções Pl/SQL, executadas no contexto do SGBD Oracle.

C.2 Classe de Suporte ao SI - *SIAccess*

Declaração

Esta classe implementa a interface remota *SIAccessI*, que contém os métodos definidos para o acesso ao sistema de informação

```
public class SIAccess extends UnicastRemoteObject implements SIAccessI {...}
```

Métodos de gestão de entidades e seus conteúdos

- *si_newEntity*

```
public int si_newEntity (String login, String chave) throws RemoteException;
```

  - **Descrição**:
    Cria uma nova entrada no sistema de informação para a entidade cujo *login* e chave foram passados.
  - **Parâmetros**:
    * login - identificador da nova entidade cuja entrada deve ser criada;
    * chave - chave de acesso definida para a nova entidade.
  - **Retorna**:
    Valor inteiro com indicação de operação efectuada com sucesso (valor positivo).

- *si_setEntity*

```
public int si_setEntity (String login, MGAttribs attribs) throws RemoteException;
```

  - **Descrição**:
    Actualiza os atributos da entidade cujo identificador (*login*) é passado.
  - **Parâmetros**:
    * login - identificador da entidade cujos atributos devem ser actualizados;
    * attribs - objecto com a lista de atributos a serem actualizados no SI.
  - **Retorna**:
    Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_consultEntity*

```
public MGAttribs si_consultEntity ( String login ) throws RemoteException;
```

  - **Descrição**:
    Dado o identificador de uma entidade (*login*), retorna os atributos referentes a esta, que se encontram actualmente armazenados no SI.
  - **Parâmetros**:
    * login - identificador da entidade cujos atributos se pretende consultar.
  - **Retorna**:
    Objecto com os atributos actuais que se encontram no SI, da entidade cujo *login* foi indicado como argumento.

- *si_removeEntity*

```
public int si_removeEntity (String login ) throws RemoteException;
```

  - **Descrição**:
    Dado o identificador de uma entidade (*login*), desactiva a entrada dessa entidade no SI.
  - **Parâmetros**:
    * login - identificador da entidade a desactivar.
  - **Retorna**:
    Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_updateCEntity*

```
public int si_updateCEntity ( String login, ObjApp objData ) throws RemoteException;
```

  - **Descrição**:
    Dado o identificador de uma entidade *login*, cria entrada para o conteúdo, passado como argumento, no sistema de informação, associando-o à entidade e devolvendo o seu identificador.
  - **Parâmetros**:
    * login - identificador da entidade;
    * objData - objecto de dados a inserir no sistema de informação.
  - **Retorna**:
    Valor com identificador do conteúdo no sistema de informação.

- *si_getCEntity*

```
public ObjApp si_getCEntity ( String login, int dataId ) throws RemoteException;
```

  - **Descrição**:
    Dado o identificador de uma entidade *login* e de um conteúdo, retorna os dados do mesmo, no caso deste existir no sistema de informação.
  - **Parâmetros**:
    * login - identificador da entidade;
    * dataId - identificador do conteúdo no SI, atribuído quando da operação de *si_updateCEntity*.
  - **Retorna**:
    O objecto conteúdo com os dados do mesmo.

- *si_removeCEntity*

```
public int si_removeCEntity ( String login, Vector dataIds ) throws RemoteException;
```

– **Descrição**:
Dado o identificador de uma entidade *login* e de um vector de identificadores de conteúdos, desactiva os mesmos do SI.

– **Parâmetros**:

∗ login - identificador da entidade;

∗ dataIds - estrutura vector com identificadores de conteúdos no SI, a serem desactivados.

– **Retorna**:
Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_consultCEntity*

```
public Vector si_consultCEntity ( String login ) throws RemoteException;
```

– **Descrição**:
Dado o identificador de uma entidade *login* obtém todos os conteúdos próprios, disponibilizado por esta no SI.

– **Parâmetros**:

∗ login - identificador da entidade.

– **Retorna**:
Vector com identificadores de conteúdos associados à entidade cujo identificador foi passado como argumento.

Métodos de gestão de grupos e seus conteúdos

- *si_newGroup*

```
public int si_newGroup (String nome_grupo, String login) throws RemoteException;
```

– **Descrição**:
Cria uma nova entrada no sistema de informação para o novo grupo cujo *nome_grupo*, é passado como argumento, é também indicado o identificador da entidade responsável pela criação do grupo.

– **Parâmetros**:

∗ nome_grupo - nome do novo grupo cuja entrada deve ser criada no SI;

∗ login - identificador da entidade responsável pelo processo de criação do novo grupo.

– **Retorna**:
Valor inteiro com indicação de operação efectuada com sucesso (valor positivo).

- *si_setGroup*

```
public int si_setGroup ( String id_grupo, String nome_grupo, MGAttribs attribs )
    throws RemoteException;
```

  - **Descrição**:
    Actualiza os atributos da entidade cujo identificador (*login*) é passado.
  - **Parâmetros**:
    - id_grupo - identificador do grupo cujos atributos devem ser actualizados no SI;
    - nome_grupo - nome lógico do grupo;
    - attribs - atributos próprios do grupo a serem actualizados no SI, no caso de ser um grupo implícito são aqui inseridos os parâmetros que definem o novo grupo.
  - **Retorna**:
    Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_consultGroup*

```
public MGAttribs si_consultGroup ( String id_grupo ) throws RemoteException;
```

  - **Descrição**:
    Dado o identificador de um grupo (*id_grupo*), retorna objecto com os atributos que se encontram actualmente armazenados no SI, referentes ao grupo indicado.
  - **Parâmetros**:
    - id_grupo - identificador do grupo cujos atributos se pretendem consultar.
  - **Retorna**:
    Objecto com os atributos actuais que se encontram no SI, do grupo cujo *id_grupo* foi indicado como argumento.

- *si_removeGroup*

```
public int si_removeGroup ( String id_group ) throws RemoteException;
```

  - **Descrição**:
    Dado o identificador de um grupo *id_group*, desactiva a entrada desse grupo no SI.

  - **Parâmetros**:
    - id_group - identificador do grupo a desactivar;
  - **Retorna**:
    Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_putGroupMember*

```
public int si_putGroupMember ( String id_group, String login ) throws RemoteException;
```

  - **Descrição**:
    Dado o identificador de um grupo *id_group* e de uma entidade *login*, associa essa entidade ao grupo indicado, ou seja, filia a entidade no grupo (ao nível do SI).
  - **Parâmetros**:
    - id_group - identificador do grupo onde vai ser filiada a nova entidade;
    - login - identificador da entidade a filiar.
  - **Retorna**:
    Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_removeGroupMember*

```
public int si_removeGroupMember ( String id_group, String login ) throws
    RemoteException;
```

  - **Descrição**:
    Dado o identificador de um grupo *id_group* e de uma entidade *login*, remove a associação dessa entidade ao grupo indicado (ao nível do SI).
  - **Parâmetros**:
    - id_group - identificador do grupo de onde deve ser removida a entidade;
    - login - identificador da entidade a retirar do grupo.
  - **Retorna**:
    Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_updateCGroup*

```
public int si_updateCgroup ( String id_group, String login, ObjApp objData ) throws
    RemoteException;
```

– **Descrição**:
Dado o identificador de um grupo e de uma entidade *login*, cria entrada para o conteúdo nesse grupo no sistema de informação e devolvendo o identificador do conteúdo.

– **Parâmetros**:

∗ id_group - identificador do grupo onde vai ser colocado o conteúdo;

∗ login - identificador da entidade produtora do conteúdo;

∗ objData - objecto de dados a inserir no sistema de informação.

– **Retorna**:
Valor com identificador do conteúdo no sistema de informação.

- *si_getCGroup*

```
public ObjApp si_getCgroup ( String id_group, int dataId ) throws RemoteException;
```

– **Descrição**:
Dado o identificador de um grupo *id_group* e de um conteúdo retorna os dados do mesmo, no caso deste existir no sistema de informação.

– **Parâmetros**:

∗ id_group - identificador do grupo;

∗ dataId - identificador do conteúdo no SI, atribuído quando da operação de *si_updateCGroup*.

– **Retorna**:
O objecto conteúdo com os dados do mesmo.

- *si_removeCGroup*

```
public int si_removeCGroup ( String id_group, Vector dataIds ) throws RemoteException;
```

– **Descrição**:
Dado o identificador de um grupo (*id_group*) e de um vector de identificadores de conteúdos, desactiva os mesmos do SI ao nível do grupo indicado.

– **Parâmetros**:

∗ id_group - identificador do grupo;

∗ dataIds - estrutura vector com identificadores de conteúdos do grupo a serem desactivados

– **Retorna**:
Valor inteiro com indicação de operação efectuada com sucesso (valor positivo), ou com erro (valor negativo).

- *si_consultCGroup*

```
public Vector si_consultCGroup ( String id_group, String login ) throws
    RemoteException;
```

  - **Descrição**:
    Dado o identificador de um grupo *id_group* obtém todos os conteúdos, associados a este grupo no SI. No caso de ser indicado também um *login*, obtém todos os conteúdos disponibilizados pela entidade indicada no grupo.
  - **Parâmetros**:
    * id_group - identificador do grupo;
    * login - identificador da entidade.
  - **Retorna**:
    Vector com identificadores de conteúdos associados ao grupo cujo identificador foi passado como argumento ou, no caso de explicitar também um *login*, retorna o vector com os identificadores de todos os conteúdos disponibilizados pela entidade indicada.

Métodos de gestão de grupos implícitos

- *findMembers*

```
public Vector findMembers ( MGAttribs list_param ) throws RemoteException;
```

  - **Descrição**:
    Dado um objecto com o conjunto de atributos que devem ser verificados por um grupo implícito retorna uma estrutura com os identificadores de todas as entidades (não desactivadas no SI) candidatas a membro do grupo, ou seja que satisfazem o conjunto de condições indicadas como argumento.
  - **Parâmetros**:
    * list_param - conjunto de atributos/valor que as entidades candidatas devem satisfazer.
  - **Retorna**:
    Vector com identificadores das entidades candidatas a membros no novo grupo implícito.

- *removeImplicits*

```
public Vector removeImplicits ( String login, MGAttribs new_param ) throws
    RemoteException;
```

– **Descrição**:
Dada uma entidade (*login*) e um objecto, com um conjunto dos novos atributos definidos para a entidade, retorna estrutura com lista de identificadores dos grupos implícitos dos quais a entidade deve ser removida.

– **Parâmetros**:

∗ login - identificador da entidade cuja filiação nos grupos implícitos está a ser processada;

∗ new_param - conjunto de novos atributos/valor da entidade, que resultaram de um processo de actualização de valores.

– **Retorna**:
Vector com identificadores dos grupos dos quais a entidade deve ser removida.

- *filiateImplicits*

```
public Vector filiateImplicits ( String login, MGAttribs new_param ) throws
    RemoteException;
```

– **Descrição**:
Dada uma entidade (*login*) e um objecto, com um conjunto dos novos atributos definidos para a entidade, retorna estrutura com lista de identificadores dos grupos implícitos nos quais a entidade deve ser filiada.

– **Parâmetros**:

∗ login - identificador da entidade cuja filiação nos grupos implícitos está a ser processada;

∗ new_param - conjunto de novos atributos/valor da entidade, que resultaram de um processo de actualização de valores.

– **Retorna**:
Vector com identificadores dos grupos nos quais a entidade deve ser filiada.

C.3 Suporte à gestão de informação do SI

Modelo Entidade Relação

O modelo de dados que serve de suporte ao SI, está descrito no modelo entidade-relação representado na figura C.2.

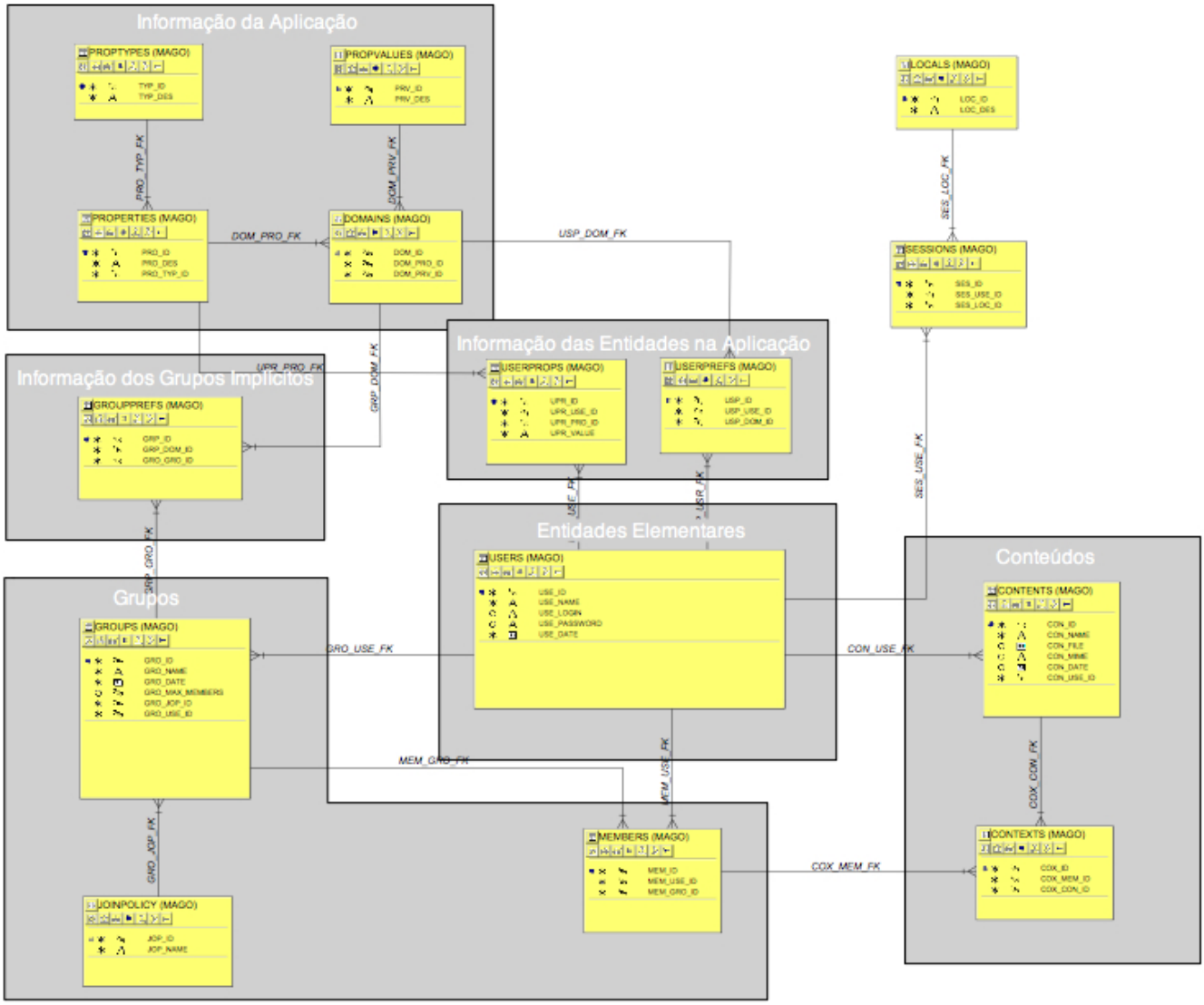

Figura C.2: Diagrama entidade-relação

Bibliografia

[AAH+97] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. In *Wireless Networks*, volume 3, pages 421–433. Springer Netherlands, October 1997.

[AAMS05] Richard Anderson, Ruth Anderson, Luke McDowell, and Beth Simon. Use of classroom presenter in engineering courses. In *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference*, pages 13–18, October 2005.

[ADKM92] Yair Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication subsystem for high availability. In *FTCS-22: 22nd International Symposium on Fault Tolerant Computing*, pages 76–84, Boston, Massachusetts, 1992. IEEE Computer Society Press.

[AMMS+95] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The totem single-ring ordering and membership protocol. *ACM Trans. Comput. Syst.*, 13(4):311–342, 1995.

[Ban98a] Bela Ban. Design and implementation of a reliable group communication toolkit for java. Technical report, Cornell University, 1998.

[Ban98b] Bela Ban. Javagroups - group communication patterns in java. Technical report, Cornell University, July 1998.

[Bar98] Jesús M. González Barahona. *A Communication Arquitecture for Process Groups*. PhD thesis, Universidad Politécnica de Madrid, 1998.

[Bar04] Fernanda Barbosa. *Abstrações de Programação Distribuída baseadas em Grupos: o modelo GroupLog*. PhD thesis, Universidade Nova de Lisboa, Portugal, 2004.

[BBD97] Ozalp Babaoglu, Alberto Bartoli, and Gianluca Dini. Enriched view synchrony: A programming paradigm for partitionable asynchronous distributed systems. *IEEE Transactions on Computers*, 46(6):642–658, 1997.

[BC91] Kenneth Birman and Robert Cooper. The isis project: real experience with a fault tolerant programming system. *SIGOPS Oper. Syst. Rev.*, 25(2):103–107, 1991.

[BC01] Fernanda Barbosa and José C. Cunha. A coordination language for collective agent based systems: Grouplog. *Applied Artificial Intelligence Journal -Special Issue on CoordinationModels and Languages in AI*, 15(1), January 2001.

[BDM01] Ozalp Babaoglu, Renzo Davoli, and Alberto Montresor. Group communication in partitionable systems: Specification and algorithms. *IEEE Trans. Softw. Eng.*, 27(4):308–336, 2001.

[BE07] Danah M. Boyd and Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication (JCMC)*, 13(1), October 2007.

[Ber06] Dave Berque. An evaluation of a broad deployment of dyknow software to support note taking and interaction using pen-based computers. *J. Comput. Small Coll.*, 21(6):204–216, 2006.

[Bir93] Kenneth P. Birman. The process group approach to reliable distributed computing. *Commun. ACM*, 36(12):37–53, 1993.

[Bir05] K. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer, May 2005.

[BJ87] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 123–138, New York, NY, USA, 1987. ACM Press.

[BMB+00] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri. Generic support for distributed applications. *Computer*, 33(3):68–76, 2000.

[BNP02] Lorenzo Bettini, Rocco De Nicola, and Rosario Pugliese. Klava: a java package for distributed and mobile applications. *Software - Practice and Experience*, 32(14):1365–1394, 2002.

[Boy07] Danah M. Boyd. Friendster lost stems. is myspace just a fad?. apophenia blog. "http://www.danah.org/papers/FriendsterMySpaceEssay.html", Accessed July 2007.

[CAC+05] Nuno Correia, Luis Alves, Helder Correia, Luis Romero, Carmen Morgado, Luís Soares, José C. Cunha, Teresa Romão, A. Eduardo Dias, and Joaquim A. Jorge. Instory: a system for mobile information access, storytelling and gaming activities in physical spaces. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 102–109, New York, NY, USA, 2005. ACM Press.

[CAM+05] N. Correia, Luís Alves, Tiago Martins, C. Morgado, J. C. Cunha, Teresa Romão, António Eduardo Dias, Joaquim A. Jorge, and Helena Barbas. Narrativas interactivas em dispositivos móveis. In *Proceedings of Artech 2005 - 2º Workshop Luso-Galaico de Artes Digitais*, pages 42–50. Grupo Português de Computação Gráfica / Eurographics, 2005.

[Car87] Jr. Nicholas Carriero. *Implementation of tuple space machines*. PhD thesis, Yale University Department of Computer Science, New Haven, Connecticut, 1987.

[CC06] Nuno Correia and Diogo Cabral. Interfaces for video based web lectures. In *ICALT '06: Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies*, pages 634–638, Washington, DC, USA, 2006. IEEE Computer Society.

[CC07] Diogo Cabral and Nuno Correia. Mobile and web tools for participative learning. In *IADIS, International Conference e-Learning 2007*. IADIS, 2007.

[CDK00] G Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. International Computer Science Series. Addison-Wesley, 3rd edition edition, August 2000.

[CDM+00] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA, 2000. ACM.

[CG89] Nicholas Carriero and David Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[CG01] Nicholas Carriero and David Gelernter. A computational model of everything. *Commun. ACM*, 44(11):77–81, 2001.

[CKV01] Gregory V. Chockler, Idid Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[CNF98] G. Cugola, E. Di Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 261–270, Washington, DC, USA, 1998. IEEE Computer Society.

[CNF01] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[Cus08] Jorge Custódio. Jgroupspace - suporte à programação distribuída orientada para grupos. Master's thesis, Faculdade de Ciência e Tecnologia / UNL, 2008.

[DA00] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *Proceedings of the Workshop on What, Who, Where, When and How of Context-Awaness*, New York, NY, 2000. CHI 2000 Conference on Human Factors in Computer Systems, ACM Press.

[DCME01] Nigel Davies, Keith Cheverst, Keith Mitchell, and Alon Efrat. Using and determining location in a context-sensitive tour guide. *Computer*, 34(8):35–41, 2001.

[Dia07] Ricardo Dias. Sistema para suporte de narrativas, acesso, partilha e visualização de informação multimédia. Technical report, Departamento de Informática, Faculdade de Ciências e Tecnologia, UNL, September 2007.

[DJFC07] Ricardo Dias, Rui Jesus, Rute Frias, and Nuno Correia. Mobile interface of the memoria project. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, page 904, New York, NY, USA, 2007. ACM Press.

[DM96] Danny Dolev and Dalia Malki. The transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–70, 1996.

[Edw01] W. Keith Edwards. *Core Jini*. Prentice Hall, Inc., 2nd edition, 2001.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[Eka07] Ekahau. Ekahau - home page. "http://www.ekahau.com/", Accessed May 2007.

[FF06] Elisabeth Freeman and Eric Freeman. *Head First HTML with CSS and XHTML*. O'Reilly Media, Inc., 2006.

[FHA99] Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces Principles Patterns, and Practice*. Prentice Hall, Inc., 1999.

[Fla98] David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Associates, 3rd edition edition, 1998.

[Fou06] The Apache Software Foundation. The apache http server project. "http://httpd.apache.org", Accessed June 2006.

[FZ94] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[Gas05] Matthew Gast. *802.11 Wireless Networks: The Definitive Guide, Second Edition (Definitive Guide)*. Nutshell Handbook. O'Reilly Media, Inc., second edition edition, April 2005.

[Gel85] David Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[Gen07] Inc. Geni. Geni - genealogy free family tree. "http://www.geni.com", Accessed September 2007.

[GPB+06] Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. *Java Concurrency in Practice*. Addison-Wesley, 2006.

[gt06] gigaspaces team. Gigaspaces extreme application platform. "http://www.gigaspaces.com", Accessed September 2006.

[Hay98] Mark Garland Hayden. *The ensemble system*. PhD thesis, Cornell University, January 1998.

[HH02] J. Scott Hamlin and Jennifer S. Hall. *Flash MX ActionScript: The Designer's Edge*. Sybex, 2002.

[HMM05] Bjarne Helvik, Hein Meling, and Alberto Montresor. An approach to experimentally obtain service dependability characteristics of the

Jgroup/ARM system. In *Proceedings of the 5th European Dependable Computing Conference (EDCC'05)*, Budapest, Hungary, April 2005.

[Hol03] Molly E. Holzschlag. *Cascading Style Sheets: The Designer's Edge*. Sybex, 1st edition, March 2003.

[HP02] Nick Heinle and Bill Peña. *Designing with JavaScript: creating Dynamic Web Pages*. O'Reilly Associates, second edition edition, 2002.

[Hum07] Lee Humphreys. Mobile social networks and social practice: A case study of dodgeball. *Journal of Computer-Mediated Communication (JCMC)*, 13(1), October 2007.

[Jav06] Jini specification and api archive. "`http://java.sun.com/products/jini/`", Accessed June 2006.

[JDF+07] R. Jesus, R. Dias, R. Frias, A. Abrantes, and N. Correia. Experiences while navigating in physical spaces. In *th Workshop on Multimedia Information Retrieval in 30th international ACM Information Retrieval Conference*. SIGIR, 2007.

[JGr06] A toolkit for reliable multicast communication. "`http://www.jgroups.org/java`", Accessed November 2006.

[Jgr07] The jgroup/arm project. "`http://jgroup.sourceforge.net/index.html`", Accessed April 2007.

[KBB+06] R. Kernchen, D. Bonnefoy, A. Battestini, B. Mrohs, M. Wagner, and M. Klemettinen. Context-awareness in mobilife. In *Proceedings of the 15th IST Mobile and Wireless Communication Summit*. Joint Workshop - Capturing Context and Context Aware Systems and Platform, 2006.

[Kei01] Idit Keidar. *Encyclopedia of Distributed Computing*. Kluwer Academic Publishers, 2001.

[KMD02] Allan Meng Krebs, Ivan Marsic, and Bogdan Dorohonceanu. Mobile adaptive applications for ubiquitous collaboration in heterogeneous environments. In *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 401–407, Washington, DC, USA, 2002. IEEE Computer Society.

[LC01] Bo Leuf and Ward Cunningham. *The Wiki Way: Collaboration and Sharing on Internet*. Addison-Wesley, 1st edition, April 2001.

[MC05] René Meier and Vinny Cahill. Taxonomy of distributed event-based programming systems. Minema reports, Helsinki University Computer Science Department and Helsinki Institute for Information Technology, 2005.

[McC97] Glen McCluskey. Remote method invocation: Creating distributed java-to-java applications. "`http://java.sun.com/developer/technicalArticles/RMI/CreatingApps/index.html`", October 1997.

[MCC07] Carmen Morgado, Nuno Correia, and J. C. Cunha. A group-based approach for modeling interactive mobile applications [poster]. International ACM Conference on Supporting Group Work - Group07, November 2007.

[McL07] Michael McLaughlin. *ORACLE Database 10g Express Edition PHP Web Programming*. McGraw-Hill Companies, Inc, 2007.

[Med07] MediaLab. Confectionary. "urlhttp://mf.media.mit.edu/confectionary/", Accessed July 2007.

[MK06] Chuck Musciano and Bill Kennedy. *HTML and XHTML: The Definitive Guide*. O'Reilly Media, Inc., 6 th edition, October 2006.

[MM00] Alberto Montresor and Hein Meling. Jgroup tutorial and programmer's manual. Technical Report UBLCS-2000-13, Department of Computer Science, University of Bologna, Setember 2000. Revised February 2002.

[MMSA+96] Louise E. Moser, P. M. Melliar-Smith, Deborah A. Agarwal, Ravi K. Budhia, and Colleen A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, 1996.

[MPSS06] Jani Mäntyjärvi, Fabio Paternò, Zigor Salvador, and Carmen Santoro. Scan and tilt: towards natural interaction for mobile museum guides. In *Mobile HCI*, pages 191–194, 2006.

[MS05] C. Morgado and Luís Soares. Mig21 api: multimedia interactive groups api. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 399–399. ACM Press, 2005.

[MSW03] Sergio Mena, André Schiper, and Pawell Wojciechowski. A step towards a new generation of group communication systems. Technical Report IC/2003/01, Ecole Polytechnique Fédérale de Lausanne (EPFL), School

of Computer and Communication Sciences, 1015 Lausanne, Switzerland, 2003.

[Muh02] Gero Muhl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, September 2002.

[MW98] Ronaldo Menezes and Alan Wood. Incorporating input/output operations in linda. In *HICSS '98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 7*, page 216, Washington, DC, USA, 1998. IEEE Computer Society.

[Nie00] Jakob Nielsen. *Designing Web Usability*. New Riders, 2000.

[Pas98] Mr. Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 92, Washington, DC, USA, 1998. IEEE Computer Society.

[PB02] P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, pages 611–618, 2002.

[Per02] José Orlando Roque Nascimento Pereira. *Semantically Reliable Group Communication*. PhD thesis, Universidade do Minho, Escola de Engenharia, Departamento de Informática, October 2002.

[Per06] Bruce Perry. *Ajax Hacks*. O'Reilly Media, Inc., 1st edition, March 2006.

[Pie04] Peter R. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, Computer Laboratory, Queens' College, University of Cambridge, 2004.

[Que05] Paulo Querido. *Amizades virtuais, Paixões reais - a sedução pela escrita*. Sociedade de Informação. Editora Centro Atlântico, October 2005.

[RBD01] Ohad Rodeh, Ken Birman, and Danny Dolev. The architecture and performance of security protocols in the ensemble group communication system: Using diamonds to guard the castle. *ACM Trans. Inf. Syst. Secur.*, 4(3):289–319, 2001.

[RBG+95] Robbert Van Renesse, Kenneth P. Birman, Bradford B. Glade, Katie Guo, Mark Hayden, Takako Hickey, Dalia Malki, Alex Vaysburd, and Werner Vogels. Horus: A flexible group communications system. Technical Report TR95-1500, Cornnel university, 23 1995.

[RCH+99] Injong Rhee, Shun Yan Cheung, Phillip W. Hutto, Alan T. Krantz, and Vaidy S. Sunderam. Group communication support for distributed collaboration systems. *Cluster Computing*, 2(1):3–16, 1999. In Proceedings of ICDCS (May 1998).

[RMJDFC07] Rui M. Jesus Rui M. Jesus, Ricardo Dias, Ricardo Dias, Rute Frias, and Nuno Correia. Geographic image retrieval in mobile guides. In *GIR 2007: Proceedings of the 4th ACM workshop on Geographical information retrieval*, Lisboa, November 2007. ACM.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 85–90, Dec 1994.

[Shi07] Clay Shirky. Clay shirky's writings about the internet. "`http://www.shirky.com/`", Accessed September 2007.

[Sof07] DyKnow Vision Software. Dyknow. "`http://www.dyknow.com/`", Accessed July 2007.

[SPRL07] Carmen Santoro, Fabio Paterno, Giulia Ricci, and Barbara Leporini. A multimodal mobile museum guide for all. Mobile Interaction with the Real World (MIRW 2007) -Workshop @ MobileHCI 2007, September 2007.

[TCC04] Roberto Tazzoli, Paolo Castagna, and Stefano Emilio Campanini. Towards a semantic wiki wiki web. In *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004.

[Tea07] CosmoCode Team. Wikimatrix. "`http://www.wikimatrix.org/`", Accessed September 2007.

[TR05] Sasu Tarkoma and Kimmo Raatikainen. State of the art review of distributed event systems. Minema reports, Helsinki University Computer Science Department and Helsinki Institute for Information Technology, 2005.

[TSp] Tspaces - programming interface specifications. "`http://www.almaden.ibm.com/cs/TSpaces`".

[UD07] Chris Ullman and Lucinda Dykes. *Beginning Ajax*. Wiley Publishing, Inc., 2007.

[vdL02] Peter van der Linden. *Just Java*. Java Series. Sun Microsystems, fifth edition edition, 2002.

[Vit98] R. Vitenberg. Properties of distributed group communication and their utilization. Master's thesis, Institute of Computer Science Hebrew University of Jerusalem, Jerusalem, Israel, January 1998.

[WCC04] G. C. Wells, A. G. Chalmers, and P. G. Clayton. Linda implementations in java for concurrent systems: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(10):1005–1022, 2004.

[web07a] canal*accessible. "http://www.zexe.net/barcelona", Accessed September 2007.

[web07b] Cyberguide project page. "http://www.cc.gatech.edu/fce/cyberguide/", Accessed June 2007.

[web07c] Linkedin home page. "http://www.linkedin.com/", Accessed May 2007.

[Wei99] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999.

[Wel04] George C. Wells. New and improved: Linda in java. In *PPPJ '04: Proceedings of the 3rd international symposium on Principles and practice of programming in Java*, pages 67–74. Trinity College Dublin, 2004.

[wik07a] Rpg - role-playing games. "http://pt.wikipedia.org/wiki/RPG_%28jogo%29", Accessed July 2007.

[wik07b] wikipedia. Social network. "http://en.wikipedia.org/wiki/Social_network", Accessed June 2007.

[wik07c] wikipedia. Social software. "http://en.wikipedia.org/wiki/Social_software", Accessed September 2007.

[wik07d] wikipedia. Wikipedia, the free encyclpedia. "http://en.wikipedia.org", 2007.

[WMLF98] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. Tspaces. *IBM Systems Journal*, 3(37), April 1998.

[YWLC99] David C. Yen, H. Joseph Wen, Binshan Lin, and David C. Chou. Groupware: a strategic analysis and implementation. *Industrial Management and Data Systems*, 99:64–70(7), February 1999.