vIBIS

Um Modelo de Discussão e Votação

Flávio Lenz Cesar

vIBIS
Um Modelo de Discussão e Votação

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Flávio Lenz Cesar e aprovada pela Comissão Julgadora.

Campinas, 15 de dezembro de 1995

Prof. Dr. Jacques Wainer
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

UNICAMP
BIBLIOTECA CENTRAL

UNIDADE BC
Nº CHAMADA:
V
EX 27274
667/96
C
PROC. R$ 11,00
DATA 10/04/96
Nº CPD CM,00026432-0

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Cesar, Flavio Lenz

C337v vIBIS - Um modelo de discussão e votação / Flavio Lenz Cesar - - Campinas, [S.P. :s.n.]. 1996.

Orientador : Jacques Wainer

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Ciência da Computação.

1. Votação - Automação. 2. Reuniões - Automação. 3. Sistemas de suporte de decisão. I. Wainer, Jacques. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Ciência da Computação. III. Título.

Tese de Mestrado defendida e aprovada em 15 de dezembro de 1995 pela Banca Examinadora composta pelos Profs. Drs.

Prof (a). Dr (a). HUGO TUNES

Prof (a). Dr (a). TOMASZ KOWALTOWSKI

Prof (a). Dr (a). JACQUES WAINER

vIBIS
Um Modelo de Discussão e Votação[1]

Flávio Lenz Cesar[2]

Banca Examinadora:

- Dr. Jacques Wainer[3] (Orientador)
- Dr. Hugo Fuks[4]
- Dr. Tomasz Kowaltowski[5]
- Dr. Jorge Stolfi[6]

---

[1] Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

[2] Bacharel em Computação pela Universidade Federal do Ceará.
[3] Professor MS-3 DCC/IMECC/UNICAMP
[4] Professor Assistente - Departamento de Informática, PUC-Rio
[5] Professor MS-6 DCC/IMECC/UNICAMP
[6] Professor MS-4 DCC/IMECC/UNICAMP

Dedicatória

Dedico esse trabalho aos meus pais e aos meus filhos, cuja mãe ainda estou para conhecer.

Agradecimentos

Meus sinceros agradecimentos a todos que, de alguma forma, contribuíram para que esse trabalho se realizasse, dos quais gostaria de destacar:

a Deus;

ao CNPq pelo apoio financeiro nos dois primeiros anos do curso;

ao Departamento de Ciência da Computação da UNICAMP pela qualidade de ensino;

ao Professor Jacques Wainer pela orientação, incentivo, amizade e paciência com que conduziu sua tarefa;

aos meus pais Homero e Hulda pelo apoio financeiro, emocional e acadêmico que me dedicaram durante toda a minha vida;

ao meu irmão Carlos e sua familia (Gerúsia, Davi, Cinthia e Bruno) pela acolhida inicial e a amizade dispensada nesses quase três anos de Campinas;

à minha irmã Paula e sua familia (Rubens, Daniel, Virgínia e Renato) pela amizade, incentivo e revisão da minha tese;

aos meus irmãos mais distantes (Helda, Silas, Cláudio e Erasmo) pelas saudades e palavras de incentivo;

à minha irmã Helda pela revisão final do texto;

à Mathilde e à Raimunda pelos serviços prestados com tanto carinho;

à Roberta e sua familia (Eudes, Marilene, Andréa, Cláudio e Flávio) pela generosidade e amizade com que sempre me receberam;

aos meus colegas de DCC pela amizade, companheirismo e momentos de descontração;

às nossas meninas (Adriana, Alda, Ana Paula, Beatriz, Camila, Eve, Jaqueline, Lucia, Luciane, Márcia, Maria Amélia, Marina, Mônica, Patricinha e Roberta) porque tornaram essa cidade muito mais interessante;

aos meus amigos Arick, Bill e Tiago pela mesmas razões expressadas acima;

à Dinalva, Eloísa, Natália, Márjorie, Naur, Núccio, Karen, Victor, Christina, Marcão, Verônica, Ronaldo, Pedro Rafael e a todas as pessoas da UNICAMP com as quais eu tive muito mais que um relacionamento acadêmico;

ao Oliva pela grande ajuda com o SYBASE.

Resumo

Nesta dissertação, faz-se um estudo sobre o processo de tomada de decisões dentro de um grupo de pessoas e propõe-se um modelo de *groupware*, denominado **vIBIS,** para discussão e votação de questões. O modelo **vIBIS** é baseado no modelo **IBIS**, concebido na década de 70 como ferramenta de *design*, e inclui uma série de métodos de votação como dispositivo de resolução das questões discutidas, o que não está disponível no modelo **IBIS** original. Faz-se um estudo detalhado sobre os mais diversos métodos de votação, e propõe-se uma nova modelagem para suas implementações.

Também é proposto um modelo de implementação para o modelo **vIBIS**, baseado na arquitetura distribuída cliente/servidor, onde um servidor central estrutura e armazena as questões discutidas e usuários remotos fazem acesso a essas discussões a partir de qualquer ponto da ***INTERNET***. Um protocolo de comunicação entre cliente e servidor é especificado, tentando facilitar ao máximo o surgimento de novas implementações de sistemas **vIBIS**, sejam elas de servidores ou clientes (interfaces). Um protótipo do sistema foi implementado e é apresentado como exemplo para essas futuras implementações.

Abstract

In this thesis we study the decision making process within a group of persons and propose a groupware model, designated as **vIBIS**, for issue's discussion and voting. The **vIBIS** model is based on the **IBIS** model, conceived as a design tool in the seventies, and it includes a series of voting methods as a discussed issues resolving device which is not available on the original **IBIS**. We perform a detailed study of several voting methods and propose a new model for their implementation.

We also propose an implementation model of the **vIBIS** model based on the client/server distributed arquitecture, where a central server organizes and stores the discussed issues and remote users may have access to these issues from any point on the ***INTERNET***. A client/server communication protocol is specified, trying to make it easy for the appearance of new **vIBIS** implementations, either for servers or clients (interfaces). We developed a prototype of the system as an example for these future implementations.

Conteúdo

Lista de Figuras

Lista de Tabelas

Introdução

CSCW (Computer Supported Cooperative Work) é um campo da Ciência da Computação que tenta facilitar e solucionar problemas de interação entre pessoas através do uso de computadores. Trabalho cooperativo não é uma idéia nova. Atividades tipicamente sociais, tais como estudar, caçar, plantar, comer, jogar e decidir vêm sendo feitas desde os primórdios da humanidade. Logicamente, algumas dessas atividades não podem ser desenvolvidas sem uma interação direta dos indivíduos, que chamaremos de interação face-a-face. Porém, muitas delas podem (sem sombra de dúvidas) acontecer sem que haja, necessariamente, a interação direta dos participantes na tarefa, desde que seja usada uma ferramenta adequada para o desenvolvimento da mesma.

A proposta de *CSCW* é fazer com que computadores sejam o meio de contato entre as pessoas em determinadas tarefas. Para tanto tenta, técnica e sociologicamente, tornar possível, viável e vantajoso o uso do computador nesses trabalhos. Sistemas desenvolvidos com esse intuito, que são conhecidos como *Groupware Systems*, têm como preocupação dar suporte à interação entre usuários, diferentemente dos sistemas computacionais comuns, que normalmente preocupam-se com a interação dos usuários com o sistema em si. Podemos citar vários bons exemplos de *Groupware Systems* que vêm sendo amplamente utilizados com bastante sucesso, tais como Correio Eletrônico (*e-mail*), ferramentas *CASE* e até mesmo Teleconferência. O programa *Lotus Notes* pode ser citado como um *groupware system* mais palpável que se tornou um grande sucesso comercial. Por palpável, entenda-se um sistema que para ser utilizado não exige uma super-estrutura como a *INTERNET*, transmissões de sinais via satélite etc. Podemos encontrar uma exposição sobre vários *groupware systems* para computadores pessoais em [OPP88] e para sistemas de maior porte em [ROB91].

Groupware Systems podem ser concebidos para ajudar tanto a grupos do tipo face-a-face, como àqueles cujos componentes estejam distribuídos em várias localidades, visto que os sistemas podem fazer uso de comunicação, não havendo necessidade das pessoas estarem num mesmo lugar. Além disso, eventualmente, determinadas interações não precisam ser feitas necessariamente em tempo real.

Johansen [JOH91] faz uma análise de tempo e espaço dos vários tipos de interação, sumarizada na Tabela 1.

Espaço/Tempo	*Mesmo Tempo*	*Tempos Diferentes*
Mesmo Lugar	INTERAÇÃO FACE-A-FACE	INTERAÇÃO ASSÍNCRONA
Lugares Diferentes	INTERAÇÃO DISTRIBUÍDA SÍNCRONA	INTERAÇÃO DISTRIBUÍDA ASSÍNCRONA

Tabela 1 - Taxonomia de tempo e espaço de *Groupware Systems*

Groupware systems podem incorporar os quatro tipos de interação apresentados na Tabela 1. Alguns deles devem ser projetados para incorporar todos os tipos. Supondo um editor de textos multiusuário, vejamos em que situações ocorreria cada tipo de interação e que comportamento o sistema deveria ter:

1) **Interação Face-a-Face**: no mesmo lugar (e.g. num escritório), um usuário edita um texto ao mesmo tempo que um colega, na mesma sala, faz alterações no texto através de uma rede local. O sistema faria com que os dois tomassem conhecimento das ações um do outro e proveria ações de *lock* para evitar conflitos nas edições.
2) **Interação Distribuída Síncrona**: usuários trabalhando em lugares diferentes (e.g. cada um em sua casa), de maneira semelhante à do exemplo anterior. Além de prover as facilidades citadas acima, o sistema faria uso de comunicação remota.
3) **Interação Assíncrona**: no mesmo ambiente, dois ou mais usuários trabalhando sobre um mesmo texto em horários diferentes. O sistema proveria maneiras de deixar bem claro que alterações o(s) último(s) usuário(s) fez(fizeram) no texto.
4) **Interação Distribuída Assíncrona**: O mesmo exemplo anterior, em locais diferentes, fazendo uso de comunicação.

CSCW relaciona-se diretamente com outras cinco áreas da computação, consideradas áreas chaves, a saber: sistemas distribuídos, comunicação, interação homem-computador, inteligência artificial e multimídia. Todas podem contribuir bastante para o desenvolvimento de *CSCW*. Por exemplo: com o surgimento de novas

arquiteturas que agilizem a comunicação e, conseqüentemente, multimídia distribuída, a tendência é que as interações eletrônicas fiquem cada vez mais naturais e atraentes; GDSSs (Sistemas de suporte à decisão em grupo[7]) [KK88], incorporando técnicas de Inteligência Artificial, poderiam melhor organizar e agilizar o processo de tomada de decisões [CM88]. Clarence Ellis [EGR91] faz um estudo detalhado sobre o impacto dessas áreas em *CSCW*.

Além das áreas estritamente ligadas à Ciência da Computação, não se pode esquecer a contribuição das Ciências Sociais, que estudam o comportamento humano durante a realização das atividades em grupo. Sem uma análise sociológica detalhada do problema, provavelmente um *groupware system* seria um fracasso. Um grupo de pesquisas do ARC (*Augment Research Center*), no Instituto de pesquisas de Stanford, que desenvolveu um sistema colaborativo chamado NLS (*On-Line System*) [EL88], chegou à conclusão que muitas das barreiras de aceitação do sistema eram mais significativamente sociais do que técnicas. Um time de psicólogos e cientistas sociais foi agregado ao grupo e era considerado tão importante quanto o próprio corpo técnico de desenvolvedores de hardware e software. Bannon e Schmidt [BS91] denominam *groupware systems* de sistemas "técnico-sociais".

Neste trabalho estamos interessados em uma forma de interação extremamente comum entre grupos de pessoas: discussão e deliberação. Quando um grupo de pessoas precisa decidir sobre um assunto qualquer, normalmente discute sobre esse assunto e ao final chega a uma conclusão sobre que ações devem ser tomadas. Normalmente isso acontece em reuniões face-a-face, onde os participantes expõem suas idéias, algumas vezes conflitantes, e depois votam para escolher uma das alternativas lançadas. Propomos um *groupware system* que possa eventualmente substituir esse tipo de reunião, com a introdução de um modelo para estruturar a discussão, mecanismos de votação automatizada com garantia de autenticidade e privacidade e toda a parte relativa à distribuição do sistema.

Nossa dissertação está organizada em 4 capítulos. No capítulo 1, fazemos uma breve explanação sobre sistemas de discussão já existentes e apresentamos o método **IBIS** (*Issue Bases Information System*), nossa principal fonte de inspiração para a parte relativa à discussão. O capítulo 2 traz um embasamento sobre a teoria das escolhas sociais, apresentando vários métodos de votação e suas formas de funcionamento. Apresentamos também nesse capítulo uma nova modelagem para esses métodos de votação. O modelo **vIBIS** (*voting* + *IBIS*), que é a implementação de domínios de votação ao modelo **IBIS**, é apresentado no capítulo 3. No capítulo 4

[7] *Group Decision Support Systems* em inglês.

são apresentados o modelo de implementação para **vIBIS** e um protótipo dessa implementação. Finalmente, apresentamos as conclusões tiradas desse trabalho bem como as contribuições científicas que podem advir do mesmo.

1. Sistemas de Discussão

Nesse capítulo apresentaremos alguns sistemas que vêm sendo comumente utilizados como sistemas de discussão de questões. Alguns desse sistemas não foram originalmente elaborados para tal fim, mas em alguns casos podem parecer plenamente satisfatórios.

1.1 Discussão eletrônica

Estamos interessados em um sistema que possa, pelo menos eventualmente, substituir as chamadas reuniões face-a-face. Discussões eletrônicas já vêm ocorrendo em larga escala nas redes de computadores, fazendo uso de ferramentas que inclusive não foram desenvolvidas para esse fim. É muito comum assistir na *INTERNET* discussões inteiras feitas através de correio eletrônico, algumas vezes de maneira bem satisfatória. Inclusive vários *mailing lists* são criados para otimizar esse processo de discussão via *e-mail*. Os usuários remetem mensagens para o servidor de listas e este se encarrega de remeter para todos os participantes. *E-mail* não foi desenvolvido com esse intuito, portanto esse processo tem os seus problemas: as mensagens ficam replicadas no *mail box* de cada usuário, gastando memória mais do que o necessário; não existe uma estruturação da discussão; dependendo do número de usuários a quem se destinem as mensagens, a remessa pode ficar muito demorada e sobrecarregar a rede; *e-mail* é um sistema que tem as suas falhas de segurança e não pode garantir a autenticidade dos usuários remetentes de mensagens. Poderíamos dizer que, para pequenas discussões sem muita importância, esse sistema é plenamente satisfatório, mas se necessitarmos de algo mais elaborado, que ofereça a segurança e eficiência de uma reunião, devemos então partir para um sistema próprio para essa tarefa.

Existem vários sistemas computacionais desenvolvidos com esse propósito. Como exemplo de sistemas *ON-LINE* temos os programas de vídeo-conferência que transmitem os sinais de audio e vídeo na rede, e, em uma arquitetura

mais modesta, os sistemas de *group talk* que interligam vários usuários em um canal de troca de mensagens. Na verdade estamos interessados em sistemas que estruturem as discussões de uma maneira que elas possam ser *OFF-LINE*, ou seja, que possam ser executadas dentro de um ambiente onde a interação seja do tipo assíncrona distribuída. Um sistema com essa característica, bem difundido e conhecido entre os usuários da *INTERNET*, é o sistema *USENET News system* [KL86]. *News*, como é popularmente conhecido, provê toda a parte relativa à distribuição e estruturação da discussão e introduz o conceito de grupos de discussão. Os usuários que quiserem fazer acesso a uma base de dados *news* podem se inscrever livremente em grupos de discussão divididos por assunto. Cada usuário pode fazer parte de vários grupos de discussão e tem acesso às discussões neles postadas. Uma discussão *news* começa da seguinte maneira: um usuário insere uma mensagem que pode ter qualquer conteúdo, normalmente uma questão; outros usuários que virem essa mensagem podem querer fazer algum comentário a respeito ou mesmo responder à pergunta, e postam uma nova mensagem, dessa vez como uma sub-mensagem, indicando que esta se refere à anterior (isso é feito através de indentação). Dessa forma, a discussão fica estruturada em forma de árvore. Discussões podem surgir de outras discussões, assim como respostas podem surgir de outras respostas. *News* estrutura bem uma discussão e é um ótimo ambiente de propagação de informações, mas, assim como *e-mail*, não provê maneiras de garantir sigilo e autenticidade dos usuários.

1.2 O modelo IBIS

O modelo **IBIS** (*Issue Based Information System*) foi criado por Rittel em 1970 [KR70] com o propósito de ser uma metodologia de *design* para problemas complexos, mas pode ser visto como uma metodologia genérica para qualquer tipo de discussão. Rittel dizia que a solução de um problema complexo adviria fundamentalmente de uma conversa entre as pessoas envolvidas na tarefa de resolver o tal problema. Esse modelo já tem 25 anos e foi utilizado por seu autor, com bastante sucesso, em diversas situações, e.g. *design* em arquitetura, planejamento urbano e planejamento na OMS[8]. O processo de argumentação, no qual se baseia o modelo, é representado por uma rede de nós e ligações, onde os principais tipos de nós são: **Questões** (*Issues*), as perguntas a serem respondidas; **Posições** (*Positions*), as alternativas de soluções para as questões; e **Argumentos** (*Arguments*), que dão suporte ou objeção às Posições. As ligações podem ser definidas entre pares de nós. Por exemplo, uma questão pode ser generalizada (especializada, sugerida) por outra

[8] Organização Mundial de Saúde

questão, uma posição pode responder uma questão e sugerir outra. Os tipos de nós e ligações do modelo são representados na Figura 1.

Para uma melhor visualização de como se disporia uma discussão **IBIS**, analisaremos o caso exemplificado na Figura 2. A questão principal é: "Como ensinar algoritmos?". Duas posições são propostas e a cada uma delas são agregados dois argumentos. Nesse caso a segunda posição sugere uma segunda questão que vai tratar da escolha da pseudo-linguagem.

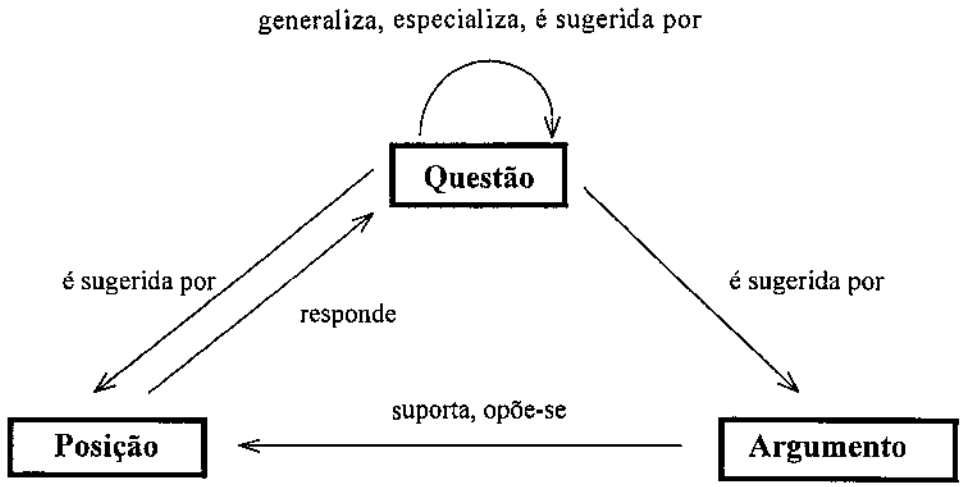


Figura 1 - Estrutura do modelo IBIS

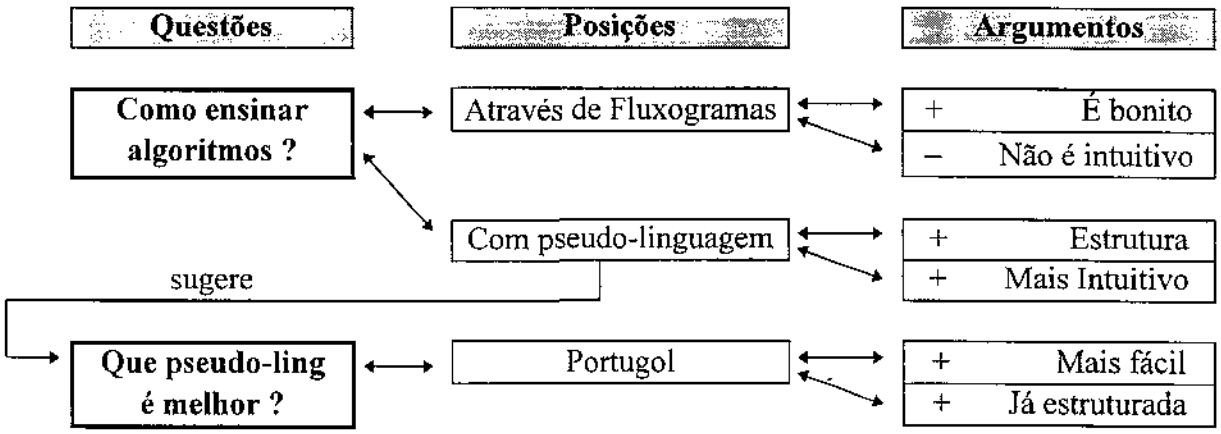


Figura 2 - Exemplo de uma discussão IBIS

A rede formada por uma questão e suas posições e argumentos relacionados é denominada *IPA* (*Issue* + *Positions* + *Arguments*). ***IPAs*** relacionam-se através de ligações questão–questão, posição–questão e argumento–questão, que

ocorrem quando um dos elementos sugere a criação de um nova questão e, conseqüentemente, de um novo *IPA*.

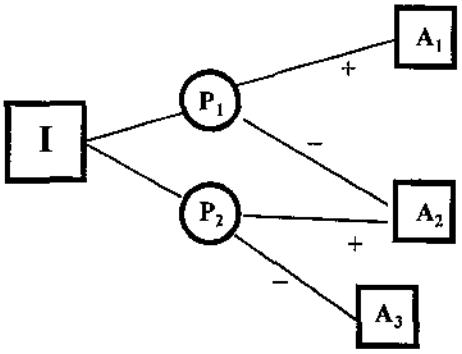


Figura 3 - Diagramação de um *IPA*

A Figura 3 mostra uma diagramação de um ***IPA*** com duas posições que respondem à questão. Argumentos podem se ligar a mais de uma posição, nesse caso o argumento A_2 é um argumento de objeção à primeira posição e de suporte à segunda. O modelo **IBIS** vem sendo utilizado largamente em *groupware systems* de discussão. **itIBIS** [YC90] implementa o modelo de uma forma textual, onde a hierarquia da sua estrutura é representada por indentação. Uma ferramenta desenvolvida na MCC [CB88] chamada **gIBIS** apresenta as redes **IBIS** em um hipertexto gráfico, fazendo

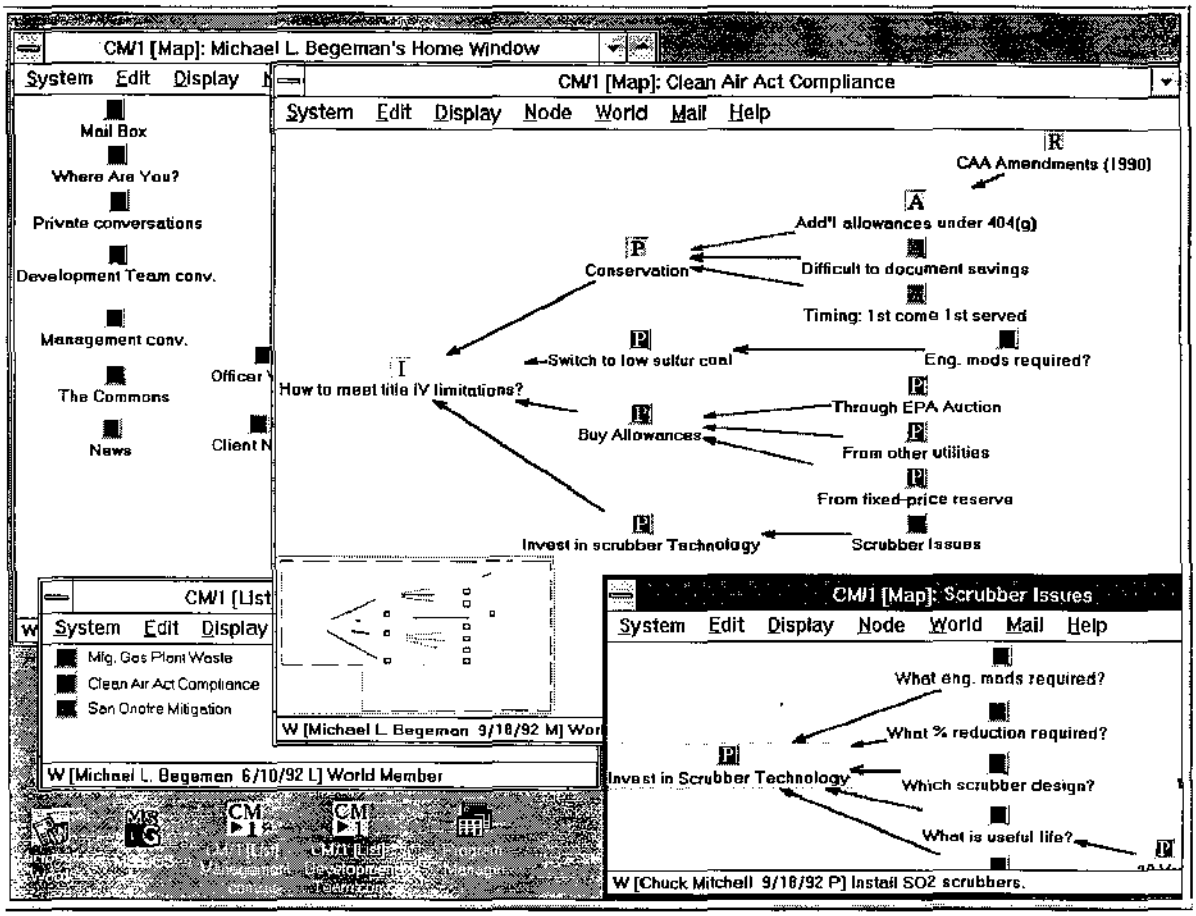


Figura 4 - *Dump* de tela do CM/1

uso de cores para melhor representar os ***IPAs*** como grafos orientados. Yakemovic e Conklin [YC90] discutem a utilização deste sistema como uma ferramenta de projeto em engenharia de software. **CM/1** [CON92] por *Corporate Memory Systems* é uma versão comercial de **gIBIS,** desenvolvido para Microsoft Windows (Figura 4). Esses sistemas, por se tratarem de implementações *off-line*, não podem ser utilizados para capturar diretamente a interação do grupo, como em uma reunião, por exemplo. Rein e Ellis [RE91] desenvolveram **rIBIS**, uma implementação em tempo real criada para esse fim.

Uma típica discussão **IBIS** inicia-se com alguém criando um nó questão contendo uma pergunta do tipo: "Como devemos fazer *X*?" Essa pessoa pode colocar também um nó posição, propondo uma alternativa de como resolver a questão e argumentar a favor dessa posição, criando nós argumentos ligados à posição. Outros participantes podem então criar vários outros nós, incrementando o conteúdo da discussão. A maneira como a discussão é estruturada deve aumentar a qualidade das discussões. Separando questões, posições e argumentos, o método tenta fazer com que fique claro para os participantes o conteúdo de cada proposta. Além disso, o sistema desencoraja ações desconstrutivas, como, por exemplo, repetir argumentações com o intuito de fazê-las mais aceitáveis. O sistema tenta também manter o foco da discussão no tema central.

O modelo vem sendo bastante experimentado nas mais diversas aplicações. Kim, Suh e Whinston [KSW93] descrevem uma extensão ao modelo para suportar discussões sobre trabalhos científicos, onde os objetos posições são redefinidos para suportar *hyphothesis*, *claims*, e *subclaims*. Esses objetos ficam organizados de forma hierárquica e podem ser ligados a argumentos de suporte ou objeção, conforme mostra a Figura 5.

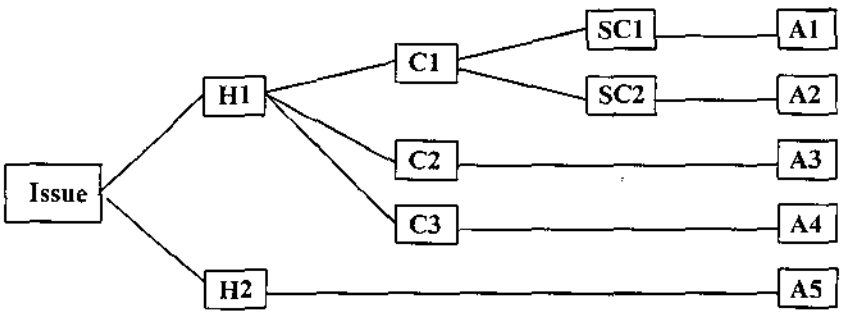


Figura 5 - Representação de um trabalho científico

Um outro modelo surgido da extensão de **IBIS** é o modelo **IBO**, que faz uso da racionalidade do modelo **IBIS** e da estruturação da metodologia de orientação a objetos para gerar um modelo de *design* de software [LL93]. Segundo os autores, as extensões feitas a partir de **IBIS** complicaram bastante o modelo, porém elas podem resolver os problemas mais precisamente. Assim como no modelo original, **IBO** não provê uma forma de finalizar uma questão.

IBIS pode ser considerado um Sistema de Suporte à Decisão (DSS)[9], mas existem alguns problemas: não existe uma regra de finalização, isto é, não existe, no modelo, uma regra particular para registrar que a discussão em torno de uma questão acaba quando os participantes da discussão chegam a um acordo sobre uma determinada posição; o modelo não provê nenhum mecanismo para chegar a uma solução.

1.3 Conclusão

Apresentamos alguns sistemas que vêm sendo comumente utilizados como sistemas de discussão de questões, entre eles, o método **IBIS**, que já vem sendo largamente utilizado como ferramenta de discussão no processo de *design*. Tal modelo se adequa perfeitamente, como sistema de discussão, ao que pretendemos em nosso trabalho final: a elaboração de um modelo de decisão que englobe discussão e votação.

[9] *Decision Support System* em inglês. DSS é uma sigla bastante difundida.

2. Sistemas de Votação

Após o processo de discussão sobre uma determinada questão, onde são apresentadas alternativas para resolvê-la, os membros do grupo de discussão precisam decidir qual ou quais alternativas serão levadas em consideração e possivelmente executadas. Isso normalmente ocorre através de uma votação, seja ela formal ou informal. O método utilizado, na maioria das vezes, é: cada membro escolhe uma alternativa e aquelas com maior número de votos são consideradas vencedoras.

Entretanto, essa forma de votação muitas vezes pode ser ineficiente, a ponto de não representar o verdadeiro desejo do grupo. O problema surge quando o grupo começa a se preocupar com a satisfação dos eleitores com o resultado da eleição. Não se pode dizer que uma alternativa, só por apresentar um maior número de votos, é a que trará maior satisfação para um grupo, nem que gerará menos insatisfação. Por exemplo, numa eleição com 2 candidatos, A e B: se A recebe maior quantidade de votos, isso apenas significa que A era desejo da maioria dos participantes, mas não significa que ele é mais consensual que B. Pode ser que A desagrade profundamente os eleitores de B, mas que B não desagrade os eleitores de A, donde B teria menos rejeição que A.

O problema se agrava quando se tem uma eleição com mais de dois candidatos. Na eleição presidencial brasileira de 1989, Collor foi eleito por maioria, porém, segundo as pesquisas de opinião, era o candidato com maior índice de rejeição. Ou seja, foi eleito o candidato que trouxe maior insatisfação para os eleitores.

Neste capítulo faremos uma breve explanação sobre eleições com múltiplos candidatos, analisaremos as mais diversas formas de votação e os diferentes métodos de seleção de vencedores. Faremos uma exposição de vários métodos clássicos descritos na literatura, sem, porém, assumir qual método é mais ou menos democrático ou consensual. Não é nossa intenção trabalhar em cima da justiça

de cada método, e sim esclarecer o seu funcionamento, para que as pessoas façam as escolhas dentro dos seus padrões de justiça e democracia.

2.1 Eleições com múltiplos candidatos

Voltando à eleição presidencial brasileira de 1989, vamos nos concentrar na pesquisa realizada por Toledo & Associados [IST89] para a revista Istoé Senhor publicada em 18 de outubro daquele ano. Considerando que a pesquisa estivesse correta, isto é, representasse a realidade das urnas, e que a eleição ocorresse naquele dia, poderíamos tirar algumas conclusões interessantes. O resultado da pesquisa trouxe, conforme mostrado na Tabela 2, Collor e Brizola como os dois mais votados, dando assim direito aos dois de disputarem o segundo turno da eleição. No segundo turno, Collor derrotaria Brizola por 48,6 a 37,4% dos votos (Tabela 3). Porém, foi feita também uma pesquisa em paralelo simulando um segundo turno entre os seis primeiros candidatos e o resultado trouxe uma surpresa: Covas seria o único candidato que derrotaria Collor (Por 47,3 a 41,9% dos votos válidos segundo a Tabela 4), e poderia derrotar também, segundo a pesquisa, todos os outros candidatos.

	Collor	Brizola	Afif	Lula	Maluf	Covas
Percentual	24,6	15,4	13	10,1	8,8	8,6

Tabela 2 - Pesquisa da eleição presidencial brasileira (1989) (Percentual de votos dos 6 primeiros)

	Brizola	Afif	Lula	Maluf	Covas
Collor	48,6 × 37,4	43 × 43	48,1 × 38	49,1 × 30,7	41,9 × 47,3

Tabela 3 - Simulação de segundo turno entre Collor e os outros candidatos

	Collor	Brizola	Afif
Covas	47,3 × 41,9	50,2 × 35,9	44,4 × 40,1

Tabela 4 - Simulação de segundo turno entre Covas e os 3 mais votados

O fato é que nada assegura que o vencedor do segundo turno de uma eleição seria o preferido em um segundo turno contra os demais candidatos. Esse tipo de eleição, em dois turnos, é feito para que cada eleitor, no primeiro turno, vote no candidato que mais lhe agrada, e, no segundo, no candidato que menos lhe desagrada.

Para tentar resolver esse tipo de problema, existem vários sistemas de votação que abrem outras possibilidades de regras para determinar um vencedor de uma eleição. Merril [MER88] analisa de perto os problemas gerados por eleições com mais de dois candidatos e vários métodos de votação, que serão apresentados neste capítulo. Faz também um estudo sobre estratégias de votação que os eleitores podem utilizar para tentar manipular o resultado de uma eleição.

2.2 Métodos de votação

Vamos supor um caso base de uma eleição com quatro alternativas (*A,B,C* e *D*), e cinco eleitores (*1,2,3,4* e *5*). No método tradicional, conhecido como votação por "Pluralidade", os eleitores *1* e *2* escolhem a alternativa *A*, o eleitor *3* prefere a alternativa *B*, o eleitor *4* vota na *C* e o eleitor *5* na alternativa *D*. Na contagem de votos, a alternativa *A* venceria com 2 votos contra 1 de cada uma das outras alternativas. Vamos supor agora que a alternativa *A* é considerada a pior por todos os outros eleitores que não *1* e *2*. O resultado da eleição desagradaria três dos cinco eleitores, ou seja, a maioria dos eleitores. Este é o tipo de problema que métodos de votação mais elaborados, e, conseqüentemente, mais complexos, tentam resolver. Dependendo de muitos fatores, que serão expostos mais adiante, o funcionamento de cada método pode gerar um resultado mais consensual ou não.

Vejamos, então, diferentes métodos de votação:

- **Pluralidade** - Este é o método mais tradicional e mais utilizado, principalmente em grandes eleições, devido à facilidade na maneira de votar e na apuração dos resultados (É o método utilizado no exemplo da Tabela 2). A cada eleitor é permitido um voto, o qual é dado, normalmente, ao candidato que mais lhe agrada. Aquele candidato que receber a maior quantidade de votos é então denominado o vencedor da eleição. Em uma eleição com somente dois cadidatos, podemos afirmar que o candidato mais votado representa a vontade da maioria, pois este obteve a maioria dos votos válidos. Porém, quando a eleição é entre três ou mais candidatos, e nenhum deles obteve a maioria dos votos, de maneira nenhuma esse método assegura que qualquer dos candidatos seja o melhor para o grupo. O que se estabelece com esse método é que o tal candidato com maior número de votos, é o que tem maior probabilidade de ser julgado pelos eleitores como superior a cada um dos outros [CON1788].

Para tentar melhorar a consensualidade do resultado das eleições, criou-se uma regra pela qual, numa situação em que nenhum dos candidatos conquiste a maioria, os dois mais votados disputam uma nova eleição, onde o vencedor é aquele com a maioria dos votos válidos. Essa é a chamada eleição em dois turnos. Dessa forma estaria assegurado que os dois candidatos que passam para o segundo turno são os com maior representatividade, mas continua não assegurando que o vencedor da eleição é o mais consensual. Um bom exemplo para mostrar isso foi a eleição para prefeito de Nova York em 1977 [BF82], onde seis candidatos obtiveram percentuais de votos entre dez e vinte por cento. Os dois mais votados obtiveram, no primeiro turno, percentuais de 19,8% e 18,6%. O problema é que nada assegura que os outros quatro candidatos não venceriam a eleição caso fossem para o segundo turno.

- ***Approval Voting*** – Uma solução apontada para tentar resolver o problema gerado por eleições por pluralidade é o método denominado *Approval Voting* [BF82], onde os eleitores votam a favor de quantos candidatos eles queiram. Dessa forma, os votos de cada eleitor indicam quais candidatos têm a sua aprovação e o vencedor é aquele que recebe o maior número de votos, ou seja, o de maior aprovação.

  No tradicional método de Pluralidade, normalmente ocorre da maioria dos eleitores ficar dividida entre os candidatos mais "fortes". Muitas vezes um eleitor deixa de votar em um candidato que gosta simplesmente pelo fato de este candidato não ter chance de ganhar a eleição e vota "útil" em um outro candidato com maiores possibilidades de vitória. Assim, o eleitor pode sentir-se melhor achando que não perdeu seu voto. Com *approval voting*, o eleitor não precisa deixar de votar em quem gosta, pois pode votar em quantos candidatos quiser. Dessa maneira, os candidatos mais "fracos" não perdem seus votos, e até podem vir a ser vitoriosos. Dentro do aspecto político, este método encoraja o surgimento de muitos partidos pequenos [BF82], já que todos dentro da eleição têm a mesma chance, independente do número de candidatos.

  Este método assegura que o ganhador da eleição é aquele que tem maior grau de aprovação entre os eleitores, porém não implica que esse candidato seja aquele que trará menos insatisfação para o grupo.

  Os custos de apuração dos votos é relativamente baixo se comparado com outros métodos mais elaborados e é facilmente

implementável em qualquer ambiente onde já existem eleições por Pluralidade. Nenhuma diferença seria necessária nas cédulas eleitorais e máquinas de contagem de votos. Uma das limitações do método é que o eleitor não pode diferenciar o quanto ele gosta de cada candidato, o que pode trazer resultados não compatíveis com a consensualidade geral do grupo.

- ***Borda* (Soma dos *rankings*)** – Jean-Charles de Borda [BOR1781] procurou uma forma alternativa ao método de pluralidade, que permitisse aos eleitores não somente manifestar seu candidato preferencial, mas também sua preferência entre todos os outros candidatos. Em uma eleição por pluralidade com três candidatos (*A*, *B* e *C*), na qual um eleitor vota em *A*, anunciando que prefere esse candidato a *B* e a *C*, nada se sabe sobre a preferência do eleitor entre os outros dois candidatos. Portanto, foi instituída a forma de voto em *ranking*, na qual o eleitor dispõe os candidatos demonstrando sua ordem de preferência. Dessa forma numa eleição com três candidatos, um eleitor expressaria seu voto por *A*>*C*>*B* indicando que prefere *A*, seguido de *C*, e, por último, do candidato *B*. O método de determinação do vencedor, denominado "soma dos *rankings*", define o vencedor da eleição como sendo:

$$\mathbf{Min} \sum_{e=1}^{u} \sum_{i=1}^{n} r_{c_i e}$$

onde u é o número de eleitores, n o número de alternativas e $r_{c_i e}$ é o *ranking* que o eleitor e associou ao candidato c_i.

Borda recomenda que se pontue cada candidato com sua posição no *ranking* de cada eleitor, e aquele que obtiver menos pontos será considerado o vencedor. Dessa maneira, considerando que um eleitor vote *"A>B>C"* expressando sua preferência por *A*, depois *B* e por último *C*, gera uma pontuação de 1 para *A*, 2 para *B* e 3 pontos para *C*. Supondo que haja mais três eleitores, e que seus votos sejam como ilustra a Tabela 5, o candidato *A* sai vitorioso, pois somou menos pontos entre os três (veja a Tabela 6). Caso a eleição fosse por pluralidade, os votos seriam *A*, *C*, *C* e *B* e o resultado traria *C* como o vencedor.

O custo computacional de um algoritmo para fazer os cálculos deste método é relativamente baixo: **O(u.n)**. Considerando que geralmente o número de eleitores é bem maior que o de candidatos, acaba ficando em **O(u)**. Porém o custo torna-se cada vez mais alto à medida que cresce o número de candidatos, e fica mais difícil de o eleitor ordenar os candidatos.

Eleitor	Voto
1	*A>B>C*
2	*C>A>B*
3	*C>A>B*
4	*B>A>C*

Tabela 5 - Votos em *Ranking*

Candidato	Pontos
A	1+2+2+2=7
B	2+3+3+1=9
C	3+1+1+3=8

Tabela 6 - Método *Borda*

- ***Condorcet*** – Marquis de Condorcet, sociólogo francês do século XVIII, empenhado em fazer utilização da matemática nas Ciências Sociais, elaborou um vasto projeto social onde enfoca principalmente o exercício da democracia através de voto. "Condorcet estabeleceu um modelo matemático baseado nas condições ideais de voto e daí concluiu uma teoria matemática das eleições tão ajustada à realidade como há duzentos anos atrás" [CON74]. A teoria argumenta que o vencedor de uma eleição deve ser o candidato que não perde de nenhum outro candidato caso a eleição fosse decidida em pequenas eleições par-a-par, decididas por pluralidade. A divisão da eleição em sub-eleições pode ser reduzida a uma única eleição com votos do tipo *ranking*, apresentado anteriormente. Como sabemos a preferência de um eleitor em relação a cada par de candidatos, sabemos também em qual candidato este eleitor votaria caso houvesse disputas entre os pares. Dessa forma, se houvesse, no exemplo da Tabela 5, mais um eleitor e seu voto fosse *C>A>B* (como mostrado na Tabela 7), *C* seria considerado o *Condorcet candidate*[10], pois pode vencer *A* e *B* por 3 votos a 2 caso fossem disputas separadas. É importante salientar que para esse caso, se o método *Borda* fosse utilizado, *A* e *C* empatariam cada um com 10 pontos.

[10] O candidato que não perde de nenhum outro, caso concorressem eleições par-a-par, é dito o *Condorcet Candidate*.

Eleitor	Voto
1	$A>B>C$
2	$C>A>B$
3	$C>A>B$
4	$B>A>C$
5	$C>A>B$

Tabela 7 - Votos em *Ranking*

Apesar desse método ser bem aceito por muitos estudiosos de sistemas de votação, ele deixa em aberto qual candidato deveria ganhar a eleição, caso não houvesse um candidato *Condorcet*, isto é, quando ocorre o chamado "Paradoxo de Votação". O fato de não haver um candidato *Condorcet* indica que para todo candidato há pelo menos um que o derrota, o que gera um ciclo de vitórias. Como exemplo clássico do paradoxo [SCH86] temos que, numa eleição com três candidatos (A,B e C) e três eleitores que votaram $A>B>C$, $B>C>A$, $C>A>B$, A ganha de B, B ganha de C e C ganha de A; ou seja, a preferência dos eleitores para mais de dois candidatos deixa de ser transitiva. DeMeyer e Plott [DP70] estudam a probabilidade de ocorrência de tais casos.

Borda, pluralidade, *Approval Voting* e outros algoritmos podem gerar vencedores que são também candidatos *Condorcet*, porém somente aqueles desenvolvidos para esse fim podem garantir se o vencedor será ou não um candidato *Condorcet*. Quanto maior for o número de eleitores, menor será a probabilidade desses algoritmos fazerem com que seus vencedores coincidam com o *Condorcet* [Mer88].

Existem alguns métodos que, assim como *Borda*, geram uma espécie de escore que pode ser utilizado tanto para determinar *Condorcet candidates* como para formar uma classificação dos candidatos. Conseqüentemente podem ser utilizados para determinar um vencedor, caso ocorra o paradoxo. Vejamos alguns desses métodos.

- ***Copeland*** – O conjunto de candidatos *Condorcet* não é obrigatoriamente unitário. Da mesma forma como esse conjunto pode ser vazio, como vimos acima, pode ocorrer também de dois candidatos empatarem entre si e ganharem de todos os outros. Nesse caso, os dois candidatos são considerados *Condorcet*. Podemos usar o "escore de *Copeland*" [MOU83] tanto para identificar candidatos *Condorcet* como para classificar os

candidatos e a partir dessa classificação determinar um vencedor. Tal escore é a contagem de quantos candidatos cada um venceria ou empataria em eleições par-a-par. Um candidato que tenha seu escore de *Copeland* maior ou igual ao escore individual de todas os outros é denominado um vencedor *Copeland*. Se o escore desse candidato for igual a $(n-1)$ onde n é o número total de candidatos, então ele também é um candidato *Condorcet*. Se o conjunto de candidatos *Condorcet* é não vazio, tal conjunto é sempre coincidente com o conjunto de vencedores Copeland.

Supondo o caso de uma eleição com quatro eleitores (*1, 2, 3 e 4*) e quatro candidatos (*A, B, C* e *D*), na qual os votos são da forma como estão dispostos na Tabela 8, construímos uma matriz 4×4 binária, onde cada elemento c_{ij} é definido como 1 se o candidato i vence ou empata com o candidato j, ou 0, se i perde de j. A soma dos elementos de cada linha i é então o número de candidatos que o elemento i pode vencer, ou seja, o escore de *Copeland*. No caso, os candidatos *B* e *C* que tiveram escore *3*, que é igual ao número de candidatos menos *1*, vencem a eleição como candidatos *Condorcet*. A classificação das alternativas, segundo esse método, fica da seguinte maneira: *B/C,A/D*.

Eleitores	Voto
1	*A>C>B>D*
2	*B>C>D>A*
3	*B>C>A>D*
4	*C>D>B>A*

Tabela 8 - Votos em *ranking*

Candidatos	*A*	*B*	*C*	*D*	Σ
A	-	0	0	1	1
B	1	-	1	1	3
C	1	1	-	1	3
D	1	0	0	-	1

Tabela 9 - Método *Copeland* para votos em *ranking*

O custo deste algoritmo é bem mais alto quando o comparamos com *Borda*, pois temos que, para cada par de alternativas, comparar todos os votos. A complexidade fica da ordem de $\mathbf{O}(u.n^2)$.

- ***Black*** – O método conhecido como *Agenda Setting Rule* apresentado por Duncan Black criou uma maneira semelhante para classificar as alternativas. Nesse método contam-se quantos votos teria cada alternativa, em cada disputa par-a-par [Bui87]. No método de *Copeland*, interessa o número de vitórias, no de *Black*, interessa o placar da vitória. No caso

apresentado acima, *B* venceria *D* por três votos a um. Dessa forma, somaria 3 pontos para *B* e 1 para *D*. O resultado final seria *C,B,A,D*, conforme ilustrado na Tabela 10.

Black não tem nenhum mecanismo para determinar candidatos *Condorcet* e pode inclusive gerar um vencedor que não é o *Condorcet*. A princípio, este método gera o mesmo resultado que o método *Borda*, pois as operações de *Black* equivalem a contar quantas alternativas estão posicionadas à direita de cada opção, enquanto *Borda* faz justamente o contrário, ou seja, conta quantas alternativas existem à esquerda (No caso, soma mais 1 ponto, o que não altera o resultado). Porém, quando for admitido igualdade de *ranking* para dois candidatos (mesma preferência do eleitor pelos dois), como veremos mais à frente, os dois métodos geram resultados diferentes. O custo computacional deste método é o mesmo apresentado por *Copeland*.

Alternativas	*A*	*B*	*C*	*D*	Σ
A	-	1	1	2	**4**
B	3	-	2	3	**8**
C	3	2	-	4	**9**
D	2	1	0	-	**3**

Tabela 10 - Método *Black* de classificação dos candidatos

- ***Kramer*** – O método *Kramer* é apresentado em [KRA77] como sendo também identificador de candidatos *Condorcet*. O modo de cálculo é bastante parecido com o método de *Black*, onde os escores das eleições par-a-par são armazenados em uma matriz. Porém, o resultado é a ordenação decrescente do menor escore obtido por cada candidato, isto é, o escore de *Kramer* é o pior resultado do candidato. O resultado da eleição apresentado na Tabela 10 seria, então, *B/C,A,D*. O escore de *Kramer* é 2 para *B* e *C* (2 foi o menor escore obtido por *B* e *C*), 1 para *A* e 0 para *D*. Os candidatos *Condorcet* são identificados quando seus escores são maiores ou iguais à metade do número de eleitores. Uma prova formal dessa afirmação pode ser vista em [Mou83].

- ***Hare*** – A regra de *Hare* [HAR1859] , também conhecida como "maioria preferencial"[11] ou "voto simples transferível"[12], assume que, se nenhum dos candidatos obtiver maioria simples[13] como primeiro colocado nos *rankings* dos votos, o candidato com menor número de votos em primeiro lugar é eliminado e uma nova contagem de primeiros colocados nos *rankings* é feita. A operação de eliminação dos candidatos menos votados se repete até que um candidato consiga a maioria. Para uma melhor ilustração deste método, vejamos a eleição onde os votos são expressos segundo a Tabela 11. Na primeira contagem, o candidato *D* é eliminado, pois foi o menos votado. Na segunda contagem, *B* é eliminado e uma nova contagem com somente os candidatos *A* e *C* é feita, sendo este último vitorioso. É importante notar que *D* é um candidato *Condorcet* e, mesmo assim, não ganhou a eleição. Para classificá-los, poderíamos usar a ordem decrescente de eliminação, resultando assim em *C,A,B,D*.

Contagem 1		Contagem 2		Contagem 3
A>B>D>C		*A>B>C*		*A>C*
A>D>B>C		*A>B>C*		*A>C*
A>D>B>C		*A>B>C*		*A>C*
B>*D>C>A*	***D* é eliminado ⇒**	***B***>*C>A*	***B* é eliminado ⇒**	***C***>*A*
B>*C>D>A*		***B***>*C>A*		***C***>*A*
C>*D>A>B*		***C***>*A>B*		***C***>*A*
C>*D>B>A*		***C***>*B>A*		***C***>*A*
D>*C>B>A*		***C***>*B>A*		***C***>*A*

Tabela 11 - Método *Hare*

Os métodos aqui apresentados são os principais citados na literatura existente sobre a Teoria das Escolhas Sociais. Tais métodos vêm sendo estudados por filósofos, matemáticos, sociólogos e economistas, que tentam estabelecer critérios para reforçar o uso de um ou outro. A escolha de um método de votação em uma eleição, muitas vezes, é um fator decisivo que pode alterar completamente o quadro da decisão. Tomemos como base as preferências dos eleitores, como as dispostas na Tabela 11, para fazer uma comparação entre os diversos métodos apresentados e mostrar como os resultados podem ser bem diferentes. Para efeito de *Approval*

[11] *Majority preferential voting* em inglês
[12] *Single transferable vote* em inglês
[13] Mais da metade dos votos

voting, consideraremos a aprovação dos eleitores como: *(A,B), (A,D,B), (A,D,B), (B), (B,C), (C), (C,D), (D,C,B).*

Comparação par-a-par					Resultado para cada método de votação					
	A	*B*	*C*	*D*	**Plurarid.**	***Approval***	***Borda***	***Copeland***	***Kramer***	***Hare***
A	-	**4**	3	3	3	3	22	1	3	3
B	**4**	-	**5**	3	2	6	20	2	3	2
C	**5**	3	-	3	2	4	21	1	3	5
D	**5**	**5**	**5**	-	1	4	17	3	5	1
Classificação:					*A,B/C,D*	*B,C/D,A*	*D,B,C,A*	*D,B,A/C*	*D,A/B/C*	*C,A,B,D*

Tabela 12 - Comparação dos diversos métodos de votação

Na Tabela acima podemos perceber que, apesar de haver algumas coincidências no vencedor, não há nenhuma coincidência na classificação dos candidatos, o que mostra que cada método pode gerar um resultado diferente, inclusive quanto ao vencedor. Alguns outros métodos de votação serão apresentados mais adiante.

2.3 Modelagem de métodos de votação

Analisando mais de perto os métodos de votação apresentados na seção anterior, podemos classificá-los segundo sua forma de votação. No método de pluralidade, o eleitor escolhe um e somente um candidato. Em *Approval Voting*, o eleitor escolhe todos os candidatos os quais têm sua aprovação. Finalmente, em todos os outros métodos apresentados, o eleitor dispõe os candidatos em uma forma de *ranking* e, a partir daí, são utilizadas diferentes formas para determinar quem será o vencedor. Como se pode perceber, para uma mesma forma de votação, podemos aplicar vários métodos para classificar os candidatos. Assim sendo, resolvemos criar nosso modelo de votação baseado em dois objetos (Figura 6): a forma de votação e o método de seleção do vencedor. Aplicando este modelo aos métodos apresentados, teríamos que **pluralidade** equivaleria à forma de votação, onde o eleitor escolhe um candidato, e ao método de seleção do vencedor, onde vence o que tem maior quantidade de votos. O mesmo método de seleção do vencedor seria utilizado para *Approval Voting*, mas a forma de votação seria diferente. É bom notar que não faz sentido falar em método de seleção do vencedor *Borda* ou *Copeland* para uma forma de votação do tipo *Approval Voting* pois tais algoritmos necessitam de votos na forma de *ranking*.

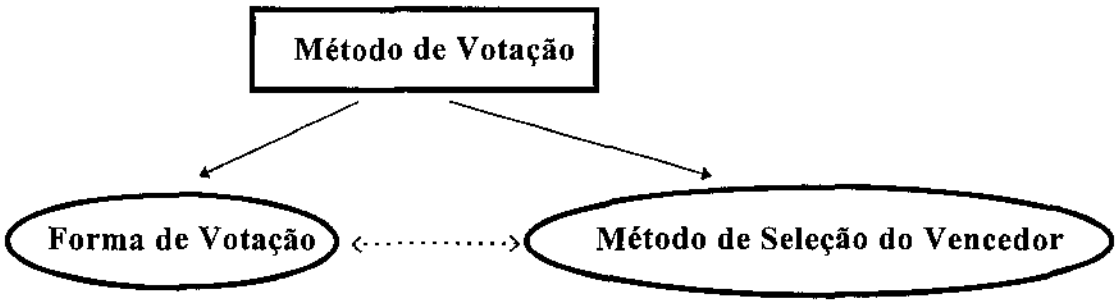


Figura 6 - Sub-divisão de um método de votação

Utilizando a divisão da Figura 6, admitimos que há uma independência relativa entre os dois objetos, tornando possível, assim, o aparecimento de novas formas de votação e novos métodos de cálculo dos vencedores. Qualquer pessoa poderia pensar, por exemplo, em um novo método que pontuasse os seis primeiros candidatos de um *ranking* do mesmo modo como são atribuídos os pontos de uma corrida de Fórmula I. Dessa forma, o primeiro receberia 10 pontos, o segundo, 6, o terceiro, 4, e 3, 2 e 1 pontos para as posições subseqüentes; o vencedor seria aquele candidato que obtivesse maior número de pontos. Stein, Mizzi e Pfaffenberger [SMP94] descrevem várias funções de pontuação para sistema de votação do tipo *ranking*. De maneira semelhante, poderíamos admitir uma nova forma de votação onde seriam permitidos dois votos para cada eleitor, o que constituiria uma extensão àquela onde é permitido somente um voto, e poderia ser representada à partir de uma parametrização da forma original, onde o parâmetro seria o número de votos admitidos para cada eleitor.

Entendemos que existe, em muitos casos, a necessidade de uma parametrização da forma de votação. Equivalentemente, pode existir algum método de seleção do vencedor que admita também uma parametrização, como, por exemplo, uma função decrescente de pontuação de *ranking*, que é uma progressão aritmética de razão r, onde r é parametrizável. A seguir, apresentaremos a nova modelagem das diferentes formas de votação e seleção de vencedores.

2.3.1 Formas de votação

Apresentamos, nessa seção, as formas de votação citadas na literatura, juntamente com as extensões propotas em nosso trabalho. Admitimos, evidentemente, a existência de idéias novas ou diferentes.

- ***Standard*** - Esta é a extensão ao tradicional método de pluralidade, onde os eleitores escolhem uma única alternativa. Como mencionado anteriormente, adicionaríamos ao método um parâmetro para permitir votos múltiplos. Dessa forma o eleitor poderia distribuir seus votos pelos diversos candidatos, e, se desejasse, poderia inclusive atribuir todos os seus votos a um candidato só. Além dessa variação, poderíamos admitir que vetos também fossem permitidos aos eleitores, de modo a não só classificar os candidatos mais representados, como também os mais rejeitados. Um método de seleção poderia levar em conta não somente o número de votos, mas também tomar o saldo de votos, ou seja, o número de votos menos o número de vetos, para classificar os candidatos. Poderíamos inclusive criar uma eleição onde o candidato eleito seria aquele que tivesse menos rejeição, permitindo que os eleitores somente manifestassem seus vetos. Generalizando tal método, o trataremos daqui por diante por método *Standard-V×S*, onde *V* é o número de votos e *S* é o número de vetos permitidos a cada eleitor. Por exemplo: *Standard-2×1* quer dizer que o número de votos permitidos para cada eleitor é 2 e o número de vetos é 1.

- ***Approval Voting*** - Para esse tipo de votação, a única mudança que desejamos fazer, apesar de desvirtuar esse método já bastante conhecido, é permitir que seja limitado o número de votos de aprovação que o eleitor pode dar. Segundo a forma tradicional, em uma eleição com 20 candidatos, o eleitor poderia aprovar todos os 20, ou seja, dar 20 votos. Segundo a extensão do método, alguém poderia escolher que o eleitor só poderia aprovar no máximo 10 candidatos. A única razão pela qual fizemos essa extensão é de tentar oferecer mais possibilidades de formas de votação. A diferença desse método para o método *Standard* é que, nesse caso, somente é permitido um voto para cada candidato. A nomenclatura para essa forma de votação passa a ser: *Approval-V*. Por exemplo: *Approval-10* para dez votos, e *Approval-0* para número indefinido de votos, ou seja, quantos candidatos existirem.

- ***Ranking*** - É a forma de votação que oferece mais possibilidades de variação nos métodos de seleção do vencedor. Originalmente, os votos seriam algo como: "*A>B>C>D*" para indicar que o eleitor gosta mais de *A*, seguido de *B, C* e por último de *D*. Porém, introduzimos o conceito de igualdade de *ranking* entre candidatos, o que permitiria que os eleitores

votassem, por exemplo, dessa forma: *"A>B=C>D"*. Nesse caso, o eleitor está indicando que a preferência de *B* é igual à de *C*, ou seja, a opinião do eleitor é indiferente aos dois candidatos. Para suportar igualdade de *ranking*, precisamos introduzir várias modificações nos métodos de pontuação que determinam a classificação dos candidatos. Veremos essas modificações na seção 2.3.2, que é dedicada a esses métodos. Em eleições com um número de candidatos muito grande, a tarefa de votar pode se tornar lenta e dispendiosa, portanto, admitimos que o eleitor possa não ter o trabalho de dispor todos os candidatos no *ranking*, o que geraria um problema no calculo das pontuações dos candidatos que não estavam no *ranking*. Nossa proposta para esses casos é considerar empatados todos os candidatos omitidos no voto, ao final do *ranking*. Por exemplo, no caso com quatro candidatos acima, se o eleitor votou somente *"A>B"*, seu voto deve ser considerado para efeito de cálculo como: *"A>B>C=D"*. Pode acontecer de, em uma eleição, ser interessante limitar o número de posições disponíveis no *ranking* do eleitor. Mais tarde veremos que métodos de seleção de vencedores podem ser criados da maneira que seu criador desejar, portanto, nada impede de um método pontuar, digamos, só os quatro primeiros candidatos do *ranking*. Assim, não faria sentido o eleitor enumerar mais que quatro candidatos em seu *ranking*. Por essa razão, incluimos um parâmetro que indica o número máximo de posições que o *ranking* pode ocupar. Similarmente a *Approval Voting*, a nomenclatura para essa forma de votação passa a ser: *Ranking-V*. Por exemplo: *Ranking-4* para o exemplo citado acima, e *Ranking-0* para *rankings* de tamanho indefinido.

- **Voto por Pesos** - É uma forma de votação que introduzida por nós, que não deixa de ser uma extensão ao método *Standard-100x0*. Nesse caso, cada eleitor tem um peso total de 100% para distribuir pelas alternativas que ele mais gosta. No exemplo apresentado acima, um voto que seria *"A>B>C>D"* poderia corresponder a *"A(50), B(25), C(15), D(10)"*. Para se chegar ao vencedor, a maneira mais intuitiva seria somar os valores obtidos individualmente e classificá-los segundo a ordem decrescente dessas somas. Utilizando essa maneira de votar, o eleitor pode expressar bem mais detalhadamente o seu grau de aprovação por cada um dos candidatos, porém esse nível de detalhe pode afetar a compreensão do método. O uso de estratégias de votação por parte dos eleitores poderia facilmente fazer com que esse método não atingisse o seu objetivo. Se

todos os eleitores pensassem: "Porque não dar 100% para o candidato que mais gosto?", o método estaria se equivalendo ao tradicional método de **pluralidade**. A utilização de métodos de *ranking* desencoraja esse tipo de estratégia por parte dos eleitores.

- **Voto por Notas** – Outra proposta nossa alternativa aos métodos tradicionais, e que, semelhantemente à forma de **pesos**, não teve analisado o impacto que pode trazer a uma eleição, é o método no qual os eleitores dão notas (por exemplo de 0 a 10 ou 0 a 100) a cada candidato. A classificação final pode ser dada pela média ou soma das notas dos candidatos. Média ou Soma geram o mesmo resultado. Esse método é bastante semelhante ao anterior, porém a diferença fundamental é que o eleitor pode dar nota máxima a dois candidatos, por exemplo. Assim como **voto por pesos** está diretamente relacionado com o método *Standard-100×0*, este método é altamente equivalente a um *Approval-0*, porém abre a possibilidade dos eleitores fazerem diferença entre os candidatos que aprovam. Se eleitores usarem a estratégia mencionada no método anterior, dando nota máxima para os candidatos que aprovam e 0 para os que desaprovam, o método pelo menos se iguala ao *Approval Voting*.

A implementação de grande parte dessas formas de votação só seria viável se houvesse uma maneira automática de contagem de votos. Não imaginamos, por exemplo, como uma eleição presidencial no Brasil possa ser feita com votos em *ranking*. A contagem não automática de votos muitas vezes inviabiliza o uso de métodos mais elaborados. Por sua vez, o uso de computadores para automatizar o processo de apuração dos votos encoraja a escolha desses outros métodos, que podem vir a ser mais consensuais que métodos mais simples como o *Standard*.

2.3.2 Métodos de seleção do vencedor

Consideremos novamente uma eleição com 4 alternativas e 5 eleitores, e supondo que o método *Standard-5×5* foi escolhido, sendo permitido a cada eleitor 5 votos e 5 vetos, os quais ele poderia distribuir livremente, inclusive votando mais de uma vez em uma única opção. O grupo de decisão deve então estabelecer os parâmetros para a apuração dos votos. O que representa um voto ou um veto? Que critério será usado para determinar um vencedor? Existem muitas opções que respondem a essas perguntas. Conforme descrito na seção anterior, uma solução intuitiva seria somar 1 ponto para cada voto e -1 para cada veto; o saldo de cada

alternativa seria o critério usado para determinar o vencedor. Mas um determinado grupo pode estabelecer que a alternativa consensual é aquela que traga menos infelicidade para os membros do grupo, dando assim maior peso aos vetos, que poderiam diminuir 2 pontos do saldo da alternativa. Ou ainda, poderia ser considerada inelegível a alternativa que tivesse maior número de vetos. Existe uma infinidade de métodos de cálculo dos vencedores, assim como existe uma infinidade de variações sobre como esses métodos de cálculo serão aplicados. Isso gera a necessidade de uma parametrização dos métodos.

Outro aspecto a ser lembrado é a possibilidade de empate nos números, que podem variar de método para método. Pode então ser adotado um critério de desempate, fazendo-se o cálculo do vencedor por um método diferente. Por exemplo, supondo que foi escolhido o método de saldo de votos e vetos para determinar o vencedor, poder-se-ia estabelecer que, no caso de empate, ganha aquela opção que tiver maior número de votos. Havendo novo empate, aquela que tivesse menor número de vetos seria a opção vencedora. Podem ser estabelecidos quantos critérios se queiram, mas mesmo assim, ainda existe a possibilidade de empate em todos eles. Nesse caso, não há solução automática que resolva. O grupo deve então procurar outras maneiras de entrar num consenso: negociar, ou simplesmente determinar alguém para dar um voto de "Minerva".

2.3.2.1 Seleção do vencedor para votação do tipo *Standard*

- ***>(Σ Votos)*** - Nesse caso, é o tradicional método de contagem de votos. O ganhador é aquele que apresenta maior número de votos.
- ***<(Σ Vetos)*** - Aqui, a intenção é que ganhe a opção menos rejeitada. Aquela que tem o menor número de vetos.
- ***>(Σ Votos – Vetos)*** - Já descrito anteriormente, este método toma como critério a diferença entre o número de votos e o número de vetos, o que é chamado de saldo de votos. Esse método pode variar no peso dado a votos e vetos. Um voto poderia contar 3 pontos e um veto diminuiria 2. Os pesos de votos e vetos entrariam como parâmetros do método.
- ***n-mais* vetadas fora** - Esse regra deve ser utilizada juntamente com uma das anteriores. Nesse caso, para efeito de classificação das opções, vale o número de votos e vetos, porém as ***n*** opções mais vetadas estariam fora da eleição. Isso faria com que as alternativas com maiores índices de rejeição ficassem inelegíveis. Essa regra poderia ser modificada para eliminar

também as opções menos votadas, garantindo assim que alternativas "fracas", isto é, com poucos votos mas que não foram muito vetadas, fiquem na frente de outras mais "fortes", que foram bem votadas e por rivalidade com outras opções, também "fortes", tiveram seu saldo prejudicado. É muito comum um eleitor vetar uma alternativa só pelo fato desta ser a principal concorrente da de sua preferência. Dessa forma, enquanto os eleitores de duas "fortes" ficam se degladiando, distribuindo votos e vetos entre as duas, pode ocorrer de uma terceira, que a princípio não aparentava ser concorrente, acabar ganhando a eleição pelo saldo de votos.

- ***n-mais* votados dentro** - Similarmente ao método anterior, em caso de eleições com múltiplos turnos, asseguraria a participação da opção mais votada para o turno seguinte. Claro que esse critério não poderia ser usado no último turno da eleição, pois isso significaria simplesmente usar o método ***>(Σ Votos)*** para decidir a eleição. Poder-se-ia também garantir a passagem de turno às alternativas menos vetadas. Na última seção deste capítulo, faremos uma explanação mais detalhada sobre eleições com múltiplos turnos.

Estabeleçamos uma eleição base[14] como exemplo: 5 eleitores votam em 4 candidatos com a forma de votação *Standard-1×0* ou *Standard-5×5* (Tabela 13). A variação no método de cálculo do vencedor está apresentada na Tabela 14. O resultado da eleição pode variar muito dependendo dos parâmetros aplicados (Tabela 15). Se mudarmos o peso de votos e vetos de (1/1) para (3/2)[15], a alternativa *B* passa à frente da alternativa *D*. Nesse caso a importância do veto passa a ser menor que a importância do voto. Se aplicássemos a regra na qual o método de classificação seria ***>(Σ votos)*** mas as duas opçõs mais vetadas estariam inelegíveis o resultado passaria de *(B,C,A/D)* para *(C,D)*, pois, apesar de *B* ter obtido uma maior votação, obteve também um grande número de vetos.

[14] Essa eleição será sempre citada daqui para frente.
[15] (3/2) indica que serão contados 3 pontos para cada voto e 2 pontos para cada veto.

Eleitor	*Standard-1×0*	*Standard-5×5*	
	voto	votos	vetos
1	*A*	*AAACC*	*BBBBB*
2	*A*	*AABCD*	*∅*
3	*B*	*BBBBC*	*AAADD*
4	*C*	*CCBBD*	*AAAAA*
5	*D*	*DDDBC*	*AAAAA*

Tabela 13 - Votos em uma eleição do tipo *Standard*

Candidato	*Standard-1×0*	*Standard-5×5*			
	Σ votos	*Σ votos*	*Σ vetos*	saldo *(1/1)*	saldo *(3/2)*
A	2	5	13	-8	-11
B	1	8	5	3	14
C	1	7	0	7	21
D	1	5	2	3	11

Tabela 14 - Contagem de votos e vetos

Método	Standard-1×0	Standard-5×5
>(Σ Votos)	*A,B/C/D*	*B,C,A/D*
<(Σ Vetos)	-	*C,D,B,A*
>(Σ Votos – ΣVetos) 1 pra 1	-	*C,B/D,A*
>(Σ Votos – ΣVetos) 3 pra 2	-	*C,B,D,A*

Tabela 15 - Resultado para diferentes métodos *Standard*

2.3.2.2 Seleção do vencedor para *Approval voting*

Para *Approval voting* só podemos considerar o grau de aprovação de cada uma das alternativas. Isso pode ser medido pelo número de votos que cada uma recebe, ou seja, quantos eleitores aprovam aquela alternativa. Esse é o método de seleção do vencedor que faz sentido quando se fala em *approval*. Com um método de votação que suportasse tanto aprovação como rejeição (*approval/desapproval voting*), poderíamos pensar em coisas semelhantes aos métodos de contagem de votos e vetos, porém não queremos desvirtuar este método já classicamente conhecido.

As Tabela 16 e Tabela 17 ilustram bem como ficaria a **eleição base** quando o método utilizado fosse *Approval voting*. É bom notar que faz falta uma diferenciação entre as alternativas aprovadas pelo eleitor. O eleitor *3*, que teria dado 4 votos a *B* e somente 1 a *C*, no método *Standard-5×0*, acabou dando o mesmo escore para as duas alternativas quando as aprovou.

Eleitor	Voto
1	AC
2	ABCD
3	BC
4	CBD
5	DBC

Tabela 16 - Voto na forma *Approval-0*

Candidato	Σ Votos
A	2
B	4
C	5
D	3
Resultado:	C,B,D,A

Tabela 17 - Resultado para *Approval Voting*

2.3.2.3 Seleção do vencedor para votos do tipo *Ranking*

Como já mostramos anteriormente, existem muitos métodos de cálculo para uma eleição com votos na forma de *ranking*. Por causa da nova definição de *ranking*, que suporta igualdade de preferência do eleitor em relação a dois ou mais candidatos, tivemos que introduzir algumas modificações nos algoritmos originais. Além desses algoritmos, gostaríamos de acrescentar alguns outros métodos citados na literatura. Como exemplo, utilizaremos sempre o caso da **eleição base**, onde os votos dos eleitores estão na forma de *ranking*, como mostra a Tabela 18.

Eleitor	Votos
1	A>C>B=D
2	A>B=C=D
3	B>C>A=D
4	C=B>D>A
5	D>B=C>A

Tabela 18 - Votos em *ranking* para a eleição base.

Os métodos de seleção do vencedor que suportam votos em *ranking* são:

- ***Borda*** – O algoritmo para cálculo dos escores de cada candidato continua o mesmo, i.e., cada candidato recebe como valor o número da posição que ocupa no *ranking*. No caso de haver uma igualdade para dois candidatos, é considerado que os dois candidatos ocupam a mesma posição e os subseqüentes recebem valores como se não houvesse a igualdade. Por exemplo: "$A>B=C>D$" resultaria em 1 ponto para o candidato A, 2 para B e C e 4 pontos para o candidato D.

- ***Copeland*** – A maioria da bibliografia que faz análise desse método considera sempre *rankings* diferentes para os candidatos, ou seja, não analisa votos do tipo *"A=B"*. Como estamos admitindo que os eleitores podem dar *rankings* iguais para dois ou mais candidatos, consideramos que, no caso de uma eleição par-a-par entre esses candidatos, o eleitor anulou seu voto. Essa é uma maneira de manifestar sua indiferença aos dois candidatos. Os procedimentos para preenchimento da matriz de pontuação continuam os mesmos.

Alternativas	*A*	*B*	*C*	*D*	Σ
A	-	0	0	1	**1**
B	1	-	1	1	**3**
C	1	1	-	1	**3**
D	1	0	0	-	**1**

Tabela 19 - *Copeland* para a eleição base

Pela Tabela 19, podemos notar que existem dois candidatos *Condorcet* para o nosso exemplo de eleição. A classificação final das alternativas pelo escore de *Copeland* ficou da seguinte maneira: *B/C,A/D*.

- ***Black*** – Nesse caso, usaremos o mesmo artifício que usamos no algoritmo anterior, ou seja, ignorar os votos com igualdade de *ranking*. Por exemplo, na **eleição base** (Tabela 18), *C* venceria *D* por três votos a um. Dessa forma, somaria 3 pontos para *C* e 1 para *D*. O resultado final seria *C,A/B,D*, conforme ilustrado na Tabela 20.

Alternativas	*A*	*B*	*C*	*D*	Σ
A	-	2	2	2	**6**
B	3	-	1	2	**6**
C	3	1	-	3	**7**
D	2	1	1	-	**4**

Tabela 20 - *Black* para a eleição base

- ***Kramer*** – Se utilizarmos este algoritmo da mesma forma como descrito originalmente, notamos pela Tabela 20 que o resultado da eleição seria *A,B/C/D* pois o escore de *Kramer* é 2 para *A* e 1 para as outras alternativas. Essa definição do escore de *Kramer* também considerou a inexistência de igualdades de *ranking*. Supondo que os votos de uma

eleição fossem: $A=B>C>D$, $A=B>D>C$, $A=B>D>C$ e $C>D>A=B$. Está bem claro que A e B são candidatos *Condorcet*, porém seus escores de *Kramer* seriam iguais a zero, pois o menor resultado em eleições par-a-par é quando os dois se enfrontam: eles empatam em 0 a 0.

A maneira de contornar esse problema é considerar não só o resultado de cada eleição par-a-par, mas também o número de votos válidos para aquele resultado. Dessa forma, a tabela de escores ficaria da seguinte maneira:

Alternativas	*A*	*B*	*C*	*D*	*Kramer score*
A	-	$^{2}/_{5}$	$^{2}/_{5}$	$^{2}/_{4}$	$^{2}/_{5}$
B	$^{3}/_{5}$	-	$^{1}/_{2}$	$^{2}/_{3}$	$^{1}/_{2}$
C	$^{3}/_{5}$	$^{1}/_{2}$	-	$^{3}/_{4}$	$^{1}/_{2}$
D	$^{2}/_{4}$	$^{1}/_{3}$	$^{1}/_{4}$	-	$^{1}/_{4}$

Tabela 21 - *Kramer* para a eleição base

Os escores passam a ser a divisão do número de votos a favor pelo número de votos total, por exemplo em $A \times B$, o resultado é 2 a 3 em 5 votos, resultando em $^{2}/_{5}$ para A e $^{3}/_{5}$ para B. O resultado final é $B/C,A,D$. Por esse método, os candidatos *Condorcet* são identificados pelos escores maiores ou iguais a $^{1}/_{2}$, indicando que o pior resultado da alternativa foi um empate. Toda vez que ocorrer um empate de 0 a 0 entre dois candidatos, o valor dos elementos correspondentes na matriz de escores de *Kramer* deve ser igual a $^{1}/_{2}$, indicando um empate entre os dois. Dessa forma estaria resolvido o caso em que A e B são candidatos *Condorcet*.

- ***Hare*** – Tivemos que alterar o método de maioria preferencial para suportar igualdade de *ranking* com a seguinte ação: Se n alternativas dividem o mesmo voto, consideraremos que cada uma recebe $1/n$ de um voto. Dessa maneira, o voto do eleitor *4* na **eleição base** corresponde, na primeira contagem, a meio voto para B e meio voto para C. Como nos mostra a Tabela 22, o primeiro candidato a ser eliminado é C com ½ voto, e o resultado final é B,A,D,C. Quando C é eliminado, os votos são reconsiderados como se este candidato não existisse, portanto B passa a ter um voto inteiro do quarto eleitor.

Contagem 1		Contagem 2		Contagem 3
A		*A*		*A*
A		*A*		*A*
B	***C* é eliminado ⇒**	*B*	***D* é eliminado ⇒**	*B*
½ *B*		*B*		*B*
½ *C*		*D*		*B*
D				

Tabela 22 - Método *Hare* para a eleição base

- ***Ranking* multiplicativo** – Esse método é bastante similar ao *Borda* (Soma dos *rankings*). A diferença é que, ao invés de somar os *rankings*, multiplica-se. Esse método objetiva dar a cada membro do grupo de eleitores mais impacto na decisão [BUI87]. O efeito multiplicativo possibilita a um indivíduo impor seu voto ou veto. Esse método também diminui a probabilidade de empates.

- **Fórmula 1** – Como a gama de opções que temos para calcular vencedores a partir dessa forma de votação em *ranking* é praticamente infinita, nós podemos utilizar qualquer parâmetro para classificar os candidatos de uma eleição. Similarmente à pontuação de uma corrida de Fórmula 1, onde os pontos de cada piloto variam de acordo com sua colocação, de primeiro a sexto lugar, recebendo 10, 6, 4, 3, 2 e 1 pontos, respectivamente, poderíamos atribuir valores para cada posição dos candidatos no voto e, inclusive, fixar um limite de posições que ganham pontos. Inspirados nesse modelo, criamos então o método que chamamos de **Fórmula 1**. A primeira alternativa de um voto somaria 10 pontos, a segunda 6, e assim por diante, até a sexta colocada no *ranking* do eleitor. A alternativa com maior número de pontos seria a vencedora. Em caso de empates ($A>B=C>D$, por exemplo), as alternativas dividiriam os pontos das posições que estivessem ocupando (*B e C* dividiriam a segunda e a terceira posições, cada um com 5 pontos). É importante notar que para esse caso, deixaríamos de utilizar a forma de votação *Ranking-0* e utilizaríamos a forma *Ranking-6*.

  Assim como "inventamos" esse método, qualquer pessoa pode criar o seu, utilizando os parâmetros que desejar. Algumas ligas esportivas americanas elegem seus melhores atletas utilizando sistemas de votação

por *ranking* e uma maneira própria de contagem de pontos para cada posição do *ranking* [SMP94].

A Tabela 23 resume e permite comparar os resultados dos diferentes métodos de seleção do vencedor para votos do tipo *ranking*, obtidos a partir do exemplo da **eleição base.**

Opção	*Borda*	*Copeland*	*Black*	*Kramer*
A	13	1	6	$^{2}/_{5}$
B	9	3	6	$^{1}/_{2}$
C	9	3	7	$^{1}/_{2}$
D	12	1	4	$^{1}/_{4}$
Final:	***B/C,D,A***	***B/C,A/D***	***C,A/B,D***	***B/C,A,D***

Opção	*Hare*	*Ranking* Multiplicativo	Fórmula 1
A	2	1×1×3×4×4=48	10+10+3,5+3+3=29,5
B	2,5	3×2×1×1×2=12	3,5+4,3+10+8+5=30,8
C	0,5	2×2×2×1×2=16	6+4,3+6+8+5=29,3
D	1	3×2×3×3×1=54	3,5+4,3+3,5+4+10=25,3
Final:	***B,A,D,C***	***B,C,A,D***	***B,A,C,D***

Tabela 23 - Comparação de métodos de cálculo para votos do tipo ranking

2.3.2.4 Seleção do vencedor para votos por pesos

Quando o tipo de votação é por pesos, o que temos a fazer para classificar as alternativas é somar os seus pesos, o que equivaleria a fazer a contagem de votos do tipo *Standard-100×0*. Podemos usar outras regras auxiliares para critério de desempate, como, por exemplo, a multiplicação dos pesos. Supondo que a **eleição base** utilizasse votos por pesos dispostos como na Tabela 24 o funcionamento da regra é mostrado na Tabela 25.

Eleitor	Voto
1	*A(60), C(40)*
2	*A(40), B(20), C(20), D(20)*
3	*B(90), C(10)*
4	*C(40), B(40), D(20)*
5	*D(70), B(15), C(15)*

Tabela 24- Votos por pesos para a eleição base

Candidato	Pontos
A	60+40=100
B	20+90+40+15=165
C	40+20+10=40+15=125
D	20+20+70=110
Final:	*B,C,D,A*

Tabela 25 - Resultado da eleição base com votos por pesos

2.3.2.5 Seleção do vencedor para votos por notas

Em uma votação por notas que os eleitores atribuem aos candidatos, parece-nos bastante intuitivo que a classificação dos candidatos se dê pelas médias aritméticas alcançadas. Ocorrendo empate, outros critérios podem ser estabelecidos através de funções estatísticas como média geométrica, mediana, etc. Supondo que os votos são como os dispostos na Tabela 26, teremos os resultados apresentados na Tabela 27.

Eleitor	Voto
1	*A(10), C(5)*
2	*A(10), B(5), C(5), D(5)*
3	*B(10), C(4)*
4	*C(8), B(8), D(5)*
5	*D(10), B(5), C(5)*

Tabela 26 - Votos por notas para a eleição base

Candidato	Notas	Média Aritmética	Média Geométrica[16]	Mediana
A	10,10,0,0,0	4	2,51	0
B	0,5,10,8,5	5,6	4,57	5
C	5,5,4,8,5	5,4	5,25	5
D	0,5,0,5,10	4	3,02	5
	Resultado:	*B,C,A/D*	*C,B,D,A*	*B/C/D,A*

Tabela 27 - Resultado para votação por notas

A escolha de um ou outro método de votação e/ou seleção de vencedores deve sempre levar em conta qual deles deixa os participantes do grupo de decisões

[16] Os zeros não são multiplicados mas são contados. $2,51 = (10\times10)^{1/5}$

mais confortáveis. Não existe nenhuma prova de que qualquer método leve, eventualmente, a uma escolha que seja mais consensual ou democrática que outra. Poderíamos citar muitos casos onde o uso de um determinado método resultaria em vencedores menos consensuais que os resultantes de outros métodos. Vale também lembrar que existe sempre a possibilidade do "voto útil", onde o eleitor vota contra a sua vontade, tentando manipular o resultado final da eleição. Não podemos jamais considerar que os votos representarão cem por cento o que pensam os eleitores, portanto, por mais consensual que pareça um método, sempre existe a possibilidade de o resultado não representar o consenso do grupo. Estratégias de votação existem e sempre irão existir em eleições. Brams, Fishburn [BF82] e Merril [Mer88] estudam o impacto dessas estratégias entre os vários métodos de votação.

2.4 Eleitores diferenciados

Analisamos até agora como seria o processo de tomada de decisão, através de votação, em grupos de pessoas onde todos os membros tivessem o mesmo poder de decisão. Nos casos vistos, considera-se que não existe nenhuma hierarquia entre os eleitores.

É bastante comum a existência de diferentes tipos de eleitores dentro de um processo democrático. Essa diferença, que pode ser por causa de conhecimento e/ou especialidade de cada membro sobre o problema a ser decidido, o poder hierárquico individual dentro da organização, ou mesmo o número de pessoas que cada um representa, pode ocasionar uma diversidade no peso dos votos desses eleitores.

Para adequar qualquer método apresentado neste trabalho, a fim de que eles suportem uma diferença de peso entre os eleitores, basta multiplicar os votos pelo peso do eleitor. Isso considerando que os pesos dos eleitores são bem determinados e, em alguns casos, números inteiros. Por exemplo, se um eleitor com peso 2 vota $A>B>C>D$, para efeito de cálculo, considera-se que existem dois votos iguais a esse.

A dificuldade está em determinar bem os pesos que serão usados. O peso pode ser a nível de eleitor ou a nível de grupo inteiro. Para melhor explicar, consideraremos o caso de uma eleição para reitor de uma universidade, que é bastante típico.

Existem três grupos de eleitores com direito a voto: Professores, Funcionários e Estudantes.

Consideremos cada um desses grupos com pesos na eleição de 50, 30 e 20 por cento, respectivamente.

- Suponhamos uma universidade com 2.000 professores, 3.000 funcionários e 10.000 alunos.
- Se todos os eleitores votassem, cada professor teria um peso de $(^{1}/_{40})\% = (^{50\%}/_{2.000})$; cada funcionário teria $(^{1}/_{100})\% = (^{30\%}/_{3.000})$ e cada estudante $(^{1}/_{500})\%=(^{20\%}/_{10.000})$. Transformando para números inteiros, seria o mesmo que dizer que cada voto de professor teria peso 25, de funcionário 10 e de estudante 2.

Poderíamos dizer que cada professor é responsável pelos seus 25 votos. Se ele faltar à eleição, estará passando um pedaço do poder dos professores para os alunos e funcionários.

Suponhamos agora que metade dos professores faltassem e que os alunos e funcionários comparecessem em massa. Dos 50 mil votos que os professores tinham direito, eles só usaram 25 mil. O total de votos foi (25+30+20 mil)=75 mil votos. O poder dos professores que antes era de $^{1}/_{2}$ passou a ser de $^{1}/_{3}$. Esse é um caso de peso de voto a nível de eleitor.

Quando o peso do voto é a nível de grupo de eleitores, não importa quantos professores faltem, eles sempre terão 50% do poder de decisão. Nesse caso, o peso dos votos varia de acordo com o número de membros do grupo que votaram. Dessa maneira, no caso de falta da metade dos professores, o peso dos votos duplicaria, passando a valer 50. Como conseqüência, os membros que votassem estariam respondendo pelos faltosos, o que poderia concentrar a decisão nas mãos de poucas pessoas. Por exemplo, caso a diretoria do DCE[17] de uma universidade incitasse os alunos a não votarem, e dessa forma, apenas as pessoas do grupo por eles liderados comparecessem à votação, o resultado da eleição na parte que coubesse aos estudantes seria facilmente manipulado.

Nos dois casos, os faltosos aumentam o poder de outras pessoas. No primeiro caso, aumentam o poder dos outros grupos e, no segundo, aumentam o poder das outras pessoas do mesmo grupo.

Lucas [LUC] dedica um capítulo inteiro sobre eleições onde os eleitores têm votos com pesos diferentes. Sua abordagem é diferente da mencionada acima, porém insatisfatória para os tipos de votação apresentados nesse capítulo pois considera somente votos do tipo "Sim ou Não". Nesse trabalho é apresentado o índice

[17] Diretório Central dos Estudantes

Banzhaf para medir o poder de decisão dos eleitores, que nem sempre coincide com o peso de cada eleitor. Tal índice é útil para analisar e projetar os sistemas de votação por pesos.

2.5 Eleições com múltiplos turnos

Uma das variações dos sistemas de votação é o caso de eleições com mais de um turno de votação, onde a cada processo há um refinamento do resultado. Para melhor ilustração de uma eleição com múltiplos turnos, descreveremos a do sistema eleitoral brasileiro, em dois turnos.

A eleição é do tipo *Standard-1×0*, sem diferença entre os eleitores, e o critério utilizado no cálculo do vencedor é o número de votos. Após a contagem de votos do primeiro turno, se nenhum candidato obtiver maioria, os dois mais votados disputam uma nova eleição, a do segundo turno, onde ganha quem tiver a maioria dos votos válidos. Nesse caso, a consensualidade estabelecida é medida pelo número de votos, ou seja, o candidato que obtiver a maioria é o vencedor. Acontece que, se no primeiro turno nenhum candidato obtiver essa maioria de votos, nenhum deles poderia ser considerado o mais consensual. Por isso, os dois que mais se aproximam do conceito de consensualidade estabelecido, partem para uma nova disputa, onde é remota a possibilidade de não haver um deles com maioria.

A eleição em dois turnos foi criada porque o método de um turno só parecia muito ineficiente quando do surgimento de vários partidos políticos. Na década de 70, no Brasil, havia somente dois partidos, o que assegurava que o vencedor era sempre expressado pela vontade da maioria dos eleitores.

Assim como temos eleições onde os dois candidatos mais votados passam para o segundo turno, poderíamos ter uma eleição com vários turnos onde o menos votado sairia da disputa e haveria um novo turno com os $(n-1)$ candidatos até que um obtivesse maioria. Existe ainda a possibilidade de ter uma eleição com vários turnos, fazendo uso de diferentes métodos de votação, onde em cada um seriam eliminados os candidatos com os piores desempenhos. Claro que esse tipo de eleição sairia muito dispendioso e cansativo, mas é uma opção para votações mais simples, como uma eleição dentro de uma empresa.

O que entendemos de eleição em vários turnos é que a cada turno pode ser eliminado um número x de candidatos através de um método de votação qualquer. Dessa forma estabelecemos, como mostra a Figura 7, que uma eleição pode ser definida em vários turnos, aos quais estarão associados métodos de votação não

obrigatoriamente coincidentes. Uma eleição, por exemplo, poderia ser disputada no primeiro turno pelo método ***Borda*** e no segundo turno, haver uma confirmação por **pluralidade**.

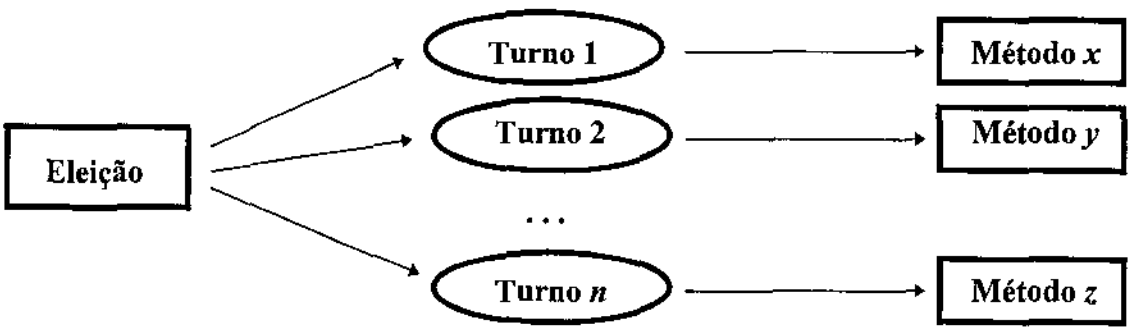


Figura 7 - Eleição em turnos

2.6 Em busca do consenso geral

Como vimos, nenhum método é cem por cento eficiente a ponto de assegurar que o vencedor de uma eleição seja o candidato mais consensual. A própria definição do que é consensual pode variar de caso a caso. Um grupo pode decidir que consensual é a vontade da maioria dos membros. Um outro grupo pode querer que seja eleito o candidato que traga menos infelicidade para os membros do grupo. O problema é que ainda não inventaram um instrumento que possa medir satisfação ou insatisfação dentro de um grupo. Podemos contar quantas pessoas estão satisfeitas e insatisfeitas, mas esse valor não necessariamente representa a satisfação total do grupo. Podem existir diferentes graus de satisfação de um indivíduo. Suponhamos uma situação em que tivéssemos 51% dos membros de um grupo satisfeitos com o resultado da eleição e os 49% restante insatisfeitos. Se os satisfeitos estivessem mediamente satisfeitos e os insatisfeitos muito insatisfeitos, e pudéssemos medir esse grau de satisfação/insatisfação, certamente que o resultado da eleição, que parecia satisfazer o grupo no geral, não poderia ser considerado satisfatório.

Esse é um problema sem solução se não houver unanimidade de opiniões. Se houver um membro qualquer que esteja muito insatisfeito, sua insatisfação pode ser maior que a soma das satisfações de todos os outros.

O que os diversos sistemas de votação procuram é oferecer opções de regras para que as pessoas possam escolher aquelas que lhes parecem mais consensuais.

2.6.1 A Teoria das Escolhas Sociais

Desde o teorema de Arrow [ARR63], primeiramente publicado em 1951, a teoria das escolhas sociais vem sendo estudada por muitos pesquisadores que tentam formular condições e axiomas para a determinação de melhores e/ou piores métodos de votação, que passaram a ser chamados, desde então, de "funções de escolha social". O estudo de Arrow é baseado na dificuldade da determinação de tais funções. O teorema estabelece que, para um universo de mais de dois candidatos, é impossível estabelecer uma função de escolha social perfeita. Para que esta função seja perfeita, Arrow assume basicamente que ela deve obedecer a cinco condições:

- **Racionalidade Coletiva** – Para qualquer conjunto de ordens (as preferências dos eleitores), o resultado da função é derivado desta ordenação.
- **Princípio do Pareto** – Se o candidato x é preferido a y por todos os indivíduos, então x também é preferido a y na ordem da sociedade.
- **Independência de Alternativas Irrelevantes** – Uma escolha social, dentro de um ambiente, deve ser dependente somente dos candidatos pertencentes àquele ambiente.
- **Não-imposição** – A sociedade pode expressar qualquer preferência entre duas alternativas.
- **Não-ditatorial** – Não existe nenhum indivíduo cuja preferência seja automaticamente a preferência da sociedade indepedente das preferências dos outros indivíduos.

Segundo o teorema da impossibilidade de Arrow, para qualquer eleição com mais de duas alternativas, não existe nenhum método de votação que satisfaça às condições acima. Explicação detalhada, prova e ilustrações para um melhor entendimento deste teorema podem ser encontrados em [Arr63] e [BLAU72].

Não discutiremos sobre a corretude ou funcionalidade do estudo de Arrow, pois tal discussão foge do escopo desta tese. Sugerimos a bibliografia ([FISH74], [KN74], [ROS75], [DDH72], [SEN77] e [Sch86]) para um aprofundamento na teoria das escolhas sociais. Em particular, gostaríamos de sugerir a leitura de [DEL91], onde o autor mostra uma refutação do teorema, e de [YOU74], que ignora parte do teorema tentando mostrar que o método *Borda* pode ser considerado "melhor" que qualquer outro.

2.7 Conclusão

Nesse capítulo fizemos uma exposição de diversos métodos de votação existentes na bibliografia e introduzimos uma nova modelagem para esses métodos, subdividindo-os em formas de votação e métodos de seleção dos vencedores. Apresentamos também as duas formas de diferenciação de eleitores, que são: pesos nos votos a nível de eleitor e pesos a nível de grupos de eleitores. Por fim, explicamos como uma eleição pode ser dividida em vários turnos de votação, cada um deles com parâmetros próprios de formas de votação e seleção dos vencedores.

3. vIBIS

Vimos nos capítulos anteriores o modelo de discussão no qual nos baseamos e diferentes métodos de votação existentes na literatura. Neste capítulo, vamos unir esses dois estudos para formar um modelo geral que englobe tanto o processo de discussão como o processo de votação em torno de uma questão.

Apesar do modelo **IBIS** não oferecer nenhum mecanismo para resolver uma questão em discussão, nós o escolhemos para compor a estrutura de discussão do nosso sistema. Resolver uma questão, parece-nos bem intuitivo, é escolher, através de um processo de votação, uma das posições que a respondem. Usando, por exemplo, um sistema de votação tradicional (pluralidade), cada participante escolhe uma posição como sendo sua opção preferida, e o sistema determina qual das posições teve o maior número de votos, sendo esta a vencedora. A extensão do modelo **IBIS** para suportar um domínio de votação nos estimulou a criar o sistema **vIBIS** (*voting* + IBIS) que modela não só a discussão e votação, mas também a forma como isso seria implementado.

Decisões, normalmente, são tomadas em reuniões face-a-face, formalmente agendadas e programadas com o propósito de discutir e/ou decidir sobre assuntos previamente divulgados. Nelas, os participantes colocam seus pontos de vista, propõem soluções para os problemas e argumentam a favor ou contra essas soluções. Os problemas com tais reuniões são muitos, entre eles:

- ausência de alguns membros do grupo;
- controle da reunião por pessoas de personalidades mais fortes;
- desvio do assunto em questão;
- escassez de tempo para os participantes apreciarem todas as opiniões e desenvolverem uma opinião antes da votação;

- simplificação dos procedimentos de votação (calculo do vencedor), já que a apuração dos votos é feita normalmente à mão;

A introdução de um sistema automático de discussão e votação resolveria muitos desses problemas[18]. Se o sistema for do tipo *off-line*, então, tanto a discussão quanto a votação podem se estender por um período grande, dando tempo para as pessoas prepararem suas posições e argumentos e considerarem seus votos. Para uma questão que não seja considerada urgente, visualizamos períodos de algumas semanas para o processo de discussão e de alguns dias para o processo de votação. O sistema poderia remeter mensagens em correio eletrônico, um tempo antes do encerramento da votação, alertando os usuários que tivessem esquecido de votar.

No aspecto da discussão, uma outra grande vantagem seria a possibilidade de contribuições anônimas. Os usuários poderiam expressar livremente suas opiniões sem medo de retaliações ou constrangimentos. A discussão não seria dominada por pessoas de personalidade mais forte, que normalmente desencorajam contra-argumentações, e o foco de uma observação levantada se concentraria no seu conteúdo e não em quem a levantou.

Um outro benefício de um sistema automático de votação é que a contagem de votos pode ser feita de maneira muito rápida, o que abre a possibilidade de resultados parciais durante a eleição, caso os membros do grupo de discussão decidissem permitir tal procedimento. Além disso, viabilizaria a utilização de qualquer método de votação. O sistema poderia permitir que os usuários reconsiderassem suas escolhas e votassem novamente, sobrepondo aos votos anteriores.

Finalmente o sistema poderia servir como registro histórico das decisões dentro do grupo. Os objetos da discussão (questões, posições, argumentos, votos, etc.) poderiam ficar armazenados de uma forma que, depois de terminado todo o processo, as pessoas pudessem recuperá-los e capturar uma parte importante do que é chamada "memória da organização" [CON92]. O sistema seria uma forma automática de alimentar esses dados relativos à memória da organização.

[18] Não estamos defendendo que todas as decisões devam ser feitas em tal sistema automático. Reuniões face-a-face servem para outros propósitos, além das decisões em si, incluindo o relacionamento social dos participantes. Cada grupo deve definir quais decisões podem ser discutidas através de uma ferramenta como essa, e quais devem ser deixadas para reuniões face-a-face.

O conceito central do modelo proposto é uma **decisão**. Uma decisão pode ter seus elementos divididos segundo dois aspectos: discussão e deliberação. Nas próximas seções analisaremos o modelo segundo esses dois aspectos.

3.1 Estrutura de discussão

Do ponto de vista da discussão, podemos dizer que uma decisão é formada por uma questão, posições que respondem a essa questão e os argumentos de suporte ou objeção ligados às posições. Entretanto nós entendemos que posições podem ser agrupadas formando sub-conjuntos com outras posições do mesmo contexto. Portanto, fizemos uma extensão ao modelo **IBIS** original, permitindo que posições especializem/generalizem outras posições, incluindo o que chamamos de "hierarquia de posições". Outros sistemas baseados no modelo **IBIS** já incorporam essa modificação, que foi proposta por [CB88]. Suponhamos uma discussão sobre a contratação de um professor de computação em uma universidade e as pessoas discutindo sobre que área da Ciência da Computação deve ser priorizada para tal contratação. Alguns propõem *Sistemas Especialistas*, outros *Algoritmos Paralelos*, *Redes Neurais*, *Processamento de Linguagem Natural* e *Algoritmos de Otimização*. Dessa forma, a discussão ficaria organizada como mostra a Figura 8. Para efeito de uma melhor organização da discussão, poderíamos incluir mais duas posições em nível mais alto que as demais: *Inteligencia Atificial* e *Teoria da Computação*, como mostra a Figura 9. Nesse caso, *IA* englobaria *SE*, *RN* e *PLN*, e Teoria da Computação englobaria as outras duas posições.

Outra mudança efetuada diz respeito ao tipo de ligação entre posições e argumentos. No modelo **IBIS** original, uma ligação argumento-posição só poderia ser

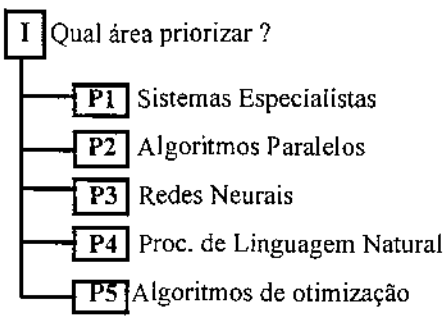


Figura 8 - Posições sem hierarquia

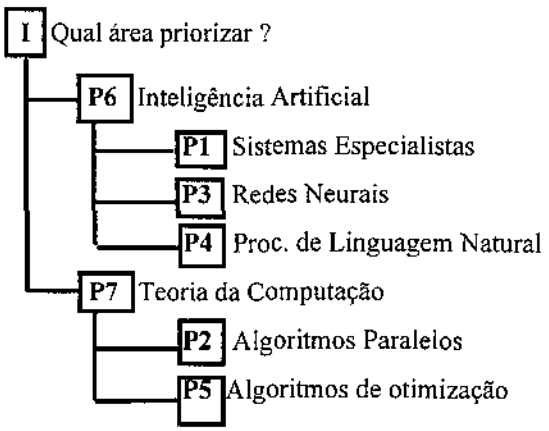


Figura 9 - Posições com hierarquia

de suporte ou objeção. Estendemos esse conceito para que argumentos sejam quaisquer comentários sobre posições. Comentários não são necessariamente de suporte ou objeção. Criamos assim um tipo de ligação entre argumentos e posições, que indica indiferença do argumento em relação à posição.

Incorporamos em **vIBIS** uma outra extensão ao **IBIS** original, que foi também inserida no sistema **gIBIS** [CB88]. É a criação de um quarto tipo de nó, que pode ser ligado a qualquer objeto de um **IPA** como uma forma de escape para armazenar objetos não-**IBIS**. Denominamos esse objeto de "Anexo" (por padronização, chamaremos de *annex*). Com esse mecanismo de escape, alguém poderia, por exemplo, fazer referência a uma comunicação interna da organização que diz respeito ao tema central da discussão e colocá-la como um *annex* ligado ao objeto que fez referência a tal.

Praticamente, qualquer sistema de discussão baseado no modelo **IBIS** poderia atender a parte relativa à discussão de uma decisão **vIBIS**. As mudanças que tais sistemas deveriam englobar seriam relativas à votação em si, bem como à cronologia da própria discussão. Por exemplo, uma discussão deve ser encerrada em algum momento, dando início ao processo de votação. Iniciado o processo de votação, nenhum dos usuários poderia fazer contribuições, ou seja, acabou a "campanha eleitoral" e não é permitido "campanha de boca de urna". Havendo um segundo turno, a discussão, se fosse retomada, só poderia se concentrar a partir das posições que tivessem obtido classificação.

Cada contribuição na discussão tem o que denominamos "autor", i.e., a pessoa que a incluiu. Esse autor pode ser divulgado ou não (contribuições anônimas), dependendo do critério estabelecido pelo grupo. A permissão para contribuições anônimas é um parâmetro da discussão que deve ser habilitado, segundo a vontade ou regra dos participantes da decisão. Uma vez habilitado esse parâmetro, o usuário escolhe se sua contribuição será anônima ou não.

3.2 Estrutura de votação

Como foi visto no capítulo anterior, uma eleição pode ocorrer de várias maneiras. Existe uma infinidade de variações, e a proposta desse trabalho é de um sistema que suporte todas, ou pelo menos a maioria dessas variações. Todas as qualidades de um sistema automático de votação, citadas no início desse capítulo, devem ser parametrizadas de forma a permitir que os participantes decidam o que,

nas suas opiniões, é mais justo e/ou democrático. Os participantes podem então decidir sobre:

- **Voto secreto**: Uma vez escolhido o voto secreto, não deve haver maneira de recuperação dos votos de qualquer usuário, nem mesmo por um super-usuário do sistema. Dessa forma, ficaria garantido o anonimato do eleitor. O sistema deve, porém, divulgar os votos, sem identificar quem os aplicou. Dessa maneira, cada usuário pode checar a existência de seu voto na apuração final. No caso de votos não secretos, o próprio sistema proveria rotinas de divulgação dos votos de cada usuário.

- **Mudança de voto**: Esse parâmetro poderia permitir que um usuário mudasse de idéia e votasse novamente, encobrindo o que tinha votado anteriormente. Logicamente que essa opção não poderia estar disponível com votos secretos, pois, nesse caso, o sistema poderia identificar o voto do usuário, o que criaria uma maneira de um super-usuário também os identificar.

- **Resultados parciais permitidos**: Uma vez permitido que resultados parciais fossem divulgados, qualquer usuário, com os devidos acessos[19], poderia pedir uma contagem de votos provisória. Tal permissão poderia fazer com que houvesse uma grande utilização de técnicas de estratégias de votação ([BF82] [MER88]), isto é, poderia haver um aumento na quantidade de votos úteis por parte das pessoas que quisessem manipular o resultado final da eleição.

- **Nós de votos**: Atrelado à extensão do modelo **IBIS**, para suportar hierarquia de posições, criamos também uma outra forma de variação da decisão. Como temos as posições estruturadas em forma de árvore, podemos permitir votos somente nos nós folhas, que contêm a especialização dos nós pais e representam um resultado final da decisão. Da mesma forma, a eleição pode ser feita por níveis, onde cada turno desceria mais nos níveis dos nós da árvore de posições. Considerando o caso da Figura 9, poderíamos ter uma eleição onde os votos deveriam ser dados para as posições folhas (*SE, RN, PLN,* etc.) e a vencedora seria a área priorizada para a contratação do professor. Outra alternativa seria a eleição em dois turnos, onde, no primeiro, seria decidido entre Inteligência Artificial e Teoria da Computação e, no segundo, qual sub-

[19] Veja parte relativa a acessos na seção sobre grupos de discussão.

ramo da área vencedora seria priorizado. No último caso teríamos uma votação por nível.

- **Número de turnos**: O número de turnos da eleição pode ser classificado como um parâmetro relativo ao aspecto da deliberação, porém, analisando mais a fundo, podemos classificá-lo como parâmetro da própria decisão, já que desse também depende a cronologia da discussão. O número de turnos de uma eleição pode ser infinito, porém deve-se tomar cuidado para não cansar o usuário com eleições sem fim. Uma eleição pode ser feita, por exemplo, de forma que cada turno elimina o candidato menos votado[20]. Dependendo do número de candidatos, essa eleição pode durar semanas ou meses. Uma decisão é então dividida em turnos, cada turno com parâmetros próprios e independentes, de períodos, métodos de votação, seleção do vencedor, etc. Passaremos então a expor esses parâmetros e o que representam:

  1. **Período de discussão**: Assim como nos sistemas políticos, deve haver, em uma decisão, um período para as pessoas fazerem suas exposições sobre o assunto em questão, ou seja, deve existir um período de "campanha eleitoral". O período de discussão inicia-se com a abertura de uma questão e estende-se até antes de começar o período de votação (normalmente, um dia antes). Durante esse período, as pessoas podem contribuir com posições, argumentos, anexos e inclusive propor sub-questões para discussões relativas à questão em pauta[21]. Como existe a possibilidade de erros de inserções, o autor de cada objeto pode removê-lo caso nenhum outro tenha sido ligado a ele e o prazo para contribuições não tenha se esgotado.

  2. **Período de votação**: Nesse parâmetro, define-se o início e o fim do período que as pessoas podem votar e, se permitido, mudar o voto e ver resultados parciais da contagem de votos. Uma vez terminado o processo de votação, os resultados podem estar disponíveis imediatamente para divulgação entre os usuários.

  3. **Forma de votação**: Como visto no Capítulo 2, existem vários métodos de votação, e cada método admite ainda uma infinidade de variações. Cada turno de uma eleição pode ser decidido por uma forma de votação. Por exemplo, podemos ter um primeiro turno com votação do

[20] Equivalente ao método *Hare* (caso os eleitores mantenham a coerência nos votos).
[21] Veja mais sobre sub-questões na seção específica.

tipo *Ranking* e um método de seleção qualquer para determinar os dois melhores candidatos (segundo esse critério). No segundo turno, pode-se estabelecer que o vencedor final é aquele com maior aprovação e aplicar-se *Approval Voting*. Cada método de votação pode ainda ter seus parâmetros. Por exemplo, o método *Standard* tem como parâmetros o número de votos e vetos permitido para cada usuário.

4. **Método de seleção do vencedor**: Dependendo da forma de votação utilizada, podemos ter várias opções de cálculo dos vencedores da eleição. Esse parâmetro deve estar de acordo com o anterior e, da mesma forma, recebe parâmetros de variação. Por exemplo, poderíamos ter uma votação do tipo *Standard* com votos e vetos, onde o peso de cada veto é maior que o peso dos votos. Esses pesos entrariam como parâmetros do método de seleção do vencedor.

5. **Método de desempate**: Como é bastante comum a existência de empates, abrimos a possibilidade da escolha de um outro método de seleção do vencedor para tratar esses casos. Se no primeiro método duas posições terminarem empatadas, o critério para a classificação seria a pontuação obtida no segundo método. Poderíamos permitir a escolha de tantos métodos de desempate quantos fossem desejados, entretanto isso carregaria demasiadamente o sistema.

6. **Número de vencedores**: Em votações com mais de um turno, deve-se estabelecer o número de posições que passarão para o turno seguinte. Dessa forma, o sistema, automaticamente, executaria a tarefa de habilitar essas posições para discussão e voto no outro turno.

3.3 Grupos de discussão

Nem sempre as reuniões estão abertas a qualquer pessoa que deseja participar delas. Normalmente, são restritas a um grupo fechado de pessoas, responsáveis pela tarefa de tomar decisões dentro de um certo domínio. Esses grupos, denominados "grupos de discussão", são formados dentro de uma organização, com o propósito de discutir e deliberar sobre determinados assuntos. Por exemplo, em um departamento de uma universidade, podem existir grupos formados só de professores, de alunos de graduação, de alunos de pós-graduação, e assim por diante. Há os casos de grupos que são formados para deliberar exclusivamente sobre um determinado assunto, por exemplo, uma comissão de professores, alunos e funcionários eleita para

resolver sobre que cor pintar o prédio do departamento. Existe ainda um tipo de reunião onde muitas pessoas podem assistir, outras podem falar, mas somente um grupo restrito pode votar. É o que chamamos de direito de voto, voz e assistência.

Além disso, diversos grupos podem ter diferentes poderes na hora da decisão. Abre-se então a possibilidade de diferença de pesos nos votos dos grupos. Por exemplo, em uma eleição para reitor de uma universidade, três grupos de pessoas podem votar, cada um deles com um peso diferente: professores, funcionários e alunos.

Modelando **vIBIS** para suportar grupos de discussão, dizemos então que, para cada questão, estão ligados vários grupos de discussão com direitos que variam de assistir a votar. No caso de direito a voto, deve constar também que peso esse grupo deve exercer, e se esse peso é a nível de grupo ou a nível de membro do grupo, como explicamos no Capítulo 2.

A princípio, um grupo deve ter condições de ser formado por qualquer usuário do sistema. Esse usuário, que cria um novo grupo de discussão, é denominado de "gerente" do grupo. O gerente tem poder para incluir e/ou excluir usuários, determinar quais domínios de ***INTERNET*** podem fazer acesso a esse grupo e permitir níveis de acesso fechado ou aberto dos domínios ao grupo. No caso de acesso fechado, os usuários do domínio devem ser incluídos pelo próprio gerente, caso contrário, qualquer usuário pertencente àquele domínio pode, ele mesmo, fazer sua inscrição e passar a ter acesso a todas as questões discutidas pelo grupo. Por exemplo, podemos ter um grupo da comunidade da UNICAMP, onde qualquer usuário de dentro da universidade pode se auto-cadastrar. Nesse caso, o gerente do grupo cadastraria todos os domínios existentes na UNICAMP (*dcc.unicamp.br, fee.unicamp.br, ifi.unicamp.br,* etc.) com acesso aberto para os seus usuários. Como existem pesquisadores da UNICAMP em outras instituições, que não fazem parte de um domínio interno, o gerente poderia dar acesso fechado a esses outros domínios (por exemplo: *cti.br, embrapa.br, cpqd.br*). Existe ainda a possibilidade de grupos acessíveis a partir de qualquer domínio.

3.4 Gerentes

Assim como em grupos de discussão, uma decisão também deve ter a figura de um gerente para controlar seus procedimentos, da mesma forma que o presidente de uma reunião a coordena. Os parâmetros de uma decisão devem e precisam ser alterados, porém não seria permitido a qualquer usuário ter acesso a

esses dados. O gerente deve ter poder total para configurar a decisão, porém fica a cargo dos participantes determinar o quanto ele pode decidir sobre essa configuração. Como a decisão será configurada deve ser decidido pelo grupo de eleitores. O gerente é só a ferramenta para a implementação das mudanças.

Nada impede que um gerente queira agir de má fé, de forma a tentar manipular o resultado de uma eleição, porém a boa organização do sistema pode inibir esse tipo de ação. O sistema poderia ser dotado de uma rotina de *e-mail* para informar aos usuários sobre as atitudes do gerente. Algumas ações poderiam ser vetadas até mesmo ao gerente após, digamos, o início de uma votação. Por exemplo, se ficou decidido que os resultados parciais não estariam disponíveis, o sistema não permitiria a mudança desse parâmetro após o início da votação.

3.5 Sub–decisões

Que métodos de votação e seleção do vencedor serão utilizados? Quantos turnos? Os votos serão secretos? Questões como essas são denominadas "questões de ordem", que em uma reunião tradicional podem paralisar todo o processo de discussão/decisão até que sejam resolvidas, pois dizem respeito à decisão em si. Os membros de uma reunião democrática normalmente têm o direito de escolher que parâmetros serão utilizados na votação de uma questão qualquer e, até que se escolham esses parâmetros, a votação não pode se iniciar.

Do ponto de vista do modelo **vIBIS**, essas sub-questões são consideradas questões inteiras, que podem conter posições, argumentos, e qualquer objeto que componha uma questão independente. Inclusive seus parâmetros, que devem também ser configurados, podem gerar sub-sub-questões para resolvê-los. Denominamos *meta-level decisions* esse tipo de decisão da qual depende o andamento de uma outra decisão. O sistema deve congelar automaticamente o processo eleitoral de uma questão até que todas suas sub-questões sejam resolvidas. Essas sub-questões devem ser levantadas sempre dentro do prazo da discussão, pois não é cabível que seja levantada uma dúvida sobre um processo de votação que já foi iniciado. Não existe nenhuma limitação de profundidade da árvore de sub-questões, mas espera-se bom senso suficiente das pessoas para não ficarem decidindo sobre o método de votação que será utilizado para decidir sobre o método de votação do método de votação, etc.

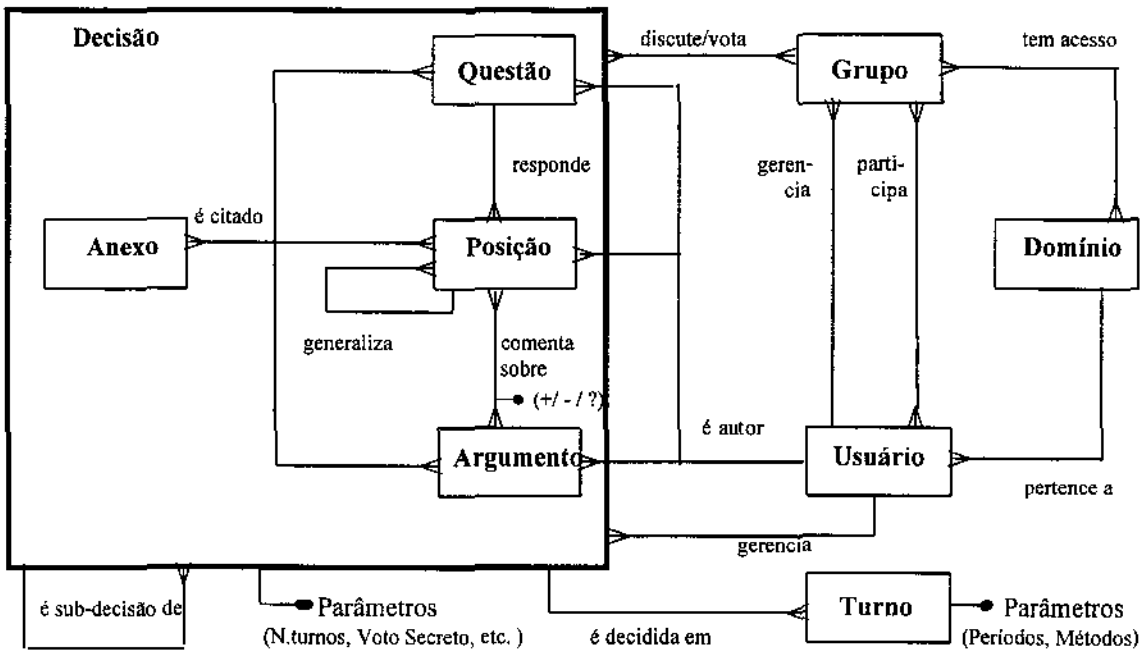


Figura 10 - Entidades-relacionamentos do modelo vIBIS

3.6 O modelo vIBIS

Na Figura 10 podemos apreciar o diagrama de entidades-relacionamentos do modelo **vIBIS**. Uma decisão, que é a composição de uma questão mais suas contribuições (posições, argumentos e anexos), pode ser uma sub-decisão de outra decisão e estar relacionada com alguns parâmetros, como número de turnos, permissão para mudança de votos, votos secretos, etc. A cada decisão estão associados seus diferentes turnos de votação, os quais também mantêm parâmetros de períodos e métodos de votação. Cada decisão é gerenciada por um usuário, que faz parte de grupos de discussão, formados por diversos usuários, que discutem/votam sobre essa decisão. Os usuários discutem sobre a questão sugerindo sub-questões, posições, argumentos e anexos que podem ter autoria ou serem anônimos, caso contribuições anônimas sejam permitidas na decisão. Cada usuário faz parte de um domínio de ***INTERNET*** e, através desse domínio, pode participar dos diversos grupos de discussão.

Internamente, o elemento principal de uma decisão é uma Questão que é respondida por várias Posições. Argumentos são comentários sobre as Posições, que podem ser de suporte, objeção ou mesmo indiferentes. Anexos são objetos que não se

enquadram como elementos **IBIS**, e que podem ser referenciados dentro de qualquer objeto de uma decisão.

3.7 Conclusão

Neste capítulo fizemos a junção de um modelo de discussão (**IBIS**) com um modelo de votação (introduzido no capítulo anterior), formando assim um modelo de discussão e votação, que foi denominado **vIBIS**. Esse modelo, além de conter algumas extensões ao modelo **IBIS** original na parte relativa à discussão, traz consigo também algumas peculiaridades como: grupos de discussão, divisão do processo decisório em turnos e parâmetros da decisão (permissão para contribuições anônimas, votos secretos, resultados parciais, etc.). Além dessa estruturação das decisões, ao modelo **vIBIS** deve ser incorporado também um modelo para sua implementação. Tal modelo de implementação será apresentado no próximo capítulo.

4. Implementação

No capítulo anterior, foi apresentada a extensão do modelo **IBIS** de discussão para suportar um domínio de votação, formando assim o modelo **vIBIS** de discussão e votação. Esse modelo foi implementado a nível de protótipo no Departamento de Ciência da Computação (**DCC**) do Instituto de Matemática, Estatística e Ciência da Computação (**IMECC**) da **UNICAMP**. Essa implementação experimental nos ajudou não somente a validar o modelo, mas também a gerar uma especificação de implementação de sistemas baseados no modelo **vIBIS**.

O sistema foi desenvolvido no ambiente **UNIX** do **DCC**, que interliga mais de 30 estações de trabalho do tipo ***SUN sparc***, 40 ***Xterminals*** e mais 60 **Machintoshes** e **PCs** que funcionam emulando terminais *X*. O **DCC** faz parte da **UNINET**, i.e., a rede local da **UNICAMP**, que permite acesso ao mundo externo através da ***INTERNET***. A rede do departamento conta ainda com um servidor **SQL** para o gerenciador de banco de dados **SYBASE**.

O sistema foi desenvolvido segundo a metodologia cliente/servidor e faz uso de comunicação via **RPC**[22] e **UNIX *pipes*** para interligar seus diferentes módulos (interface, servidor, etc.). Neste capítulo, descreveremos a modelagem de implementação de sistemas **vIBIS**, o protocolo criado para a comunicação dos processos, e o protótipo do sistema.

4.1 Modelo de implementação

O objetivo do nosso trabalho é, além de modelar a maneira como uma decisão pode ser informatizada em um computador, criar uma especificação completa de sistemas que façam uso dessa modelagem em um ambiente de computadores interligados.

[22] RPC = *Remote procedure call* - Chamada de procedimento remoto.

Devido à grande tendência de interligação dos mais diversos tipos de computadores através da rede ***INTERNET***, optamos por uma arquitetura cliente/servidor onde a interface com o usuário (que faz parte do módulo cliente) troca pacotes de **TCP** com o servidor (que faz o gerenciamento dos dados), o qual pode estar localizado em qualquer outra máquina acessível via **TCP/IP**. A intenção é abrir a possibilidade de formação de uma rede de servidores **vIBIS** pela ***INTERNET***, os quais possam receber *logins* de quaisquer usuários de quaisquer redes locais. Reuniões de decisão ocorrem em vários lugares, obrigando seus participantes a se deslocarem até os locais onde elas acontecem. Pretendemos que esses deslocamentos sejam feitos eletronicamente, através da rede, ou seja, se o usuário *X* deseja participar da discussão sobre um determinado assunto que está ocorrendo na universidade *Y*, basta se conectar ao servidor **vIBIS** daquela universidade.

A arquitetura do sistema proposto é totalmente aberta, isto é, qualquer pessoa pode implementar seus próprios módulos do sistema, desde que observe alguns padrões. Por exemplo, alguém poderia construir sua própria interface em qualquer plataforma, contanto que esta suporte **RPC**. Da mesma forma, vários servidores poderiam surgir nos mais diversos *sites* da ***INTERNET***, obedecendo à padronização e normas de segurança estabelecidas.

O esquema de disposição de clientes e servidores do nosso modelo é bastante similar ao utilizado pelo ***USENET News system***. Assim como *News* utiliza o protocolo de comunicação entre cliente e servidor **MNTP** [KL86], ele faz uso de um protocolo para suportar a comunicação entre os módulos. O protocolo está descrito no Anexo I, dessa dissertação.

Como mencionamos no capítulo 1, *News* não provê (nem precisa) nenhuma maneira de garantir a autenticidade dos usuários que se *logam* aos servidores. Qualquer *hacker*, com um mínimo de experiência de programação em baixo nível, pode postar uma mensagem em um *News Group*, simulando ser de outra pessoa. No sistema em questão, estamos lidando com informações que podem ser muito importantes para uma organização, cuja segurança é primordial para seu sucesso. Infelizmente, a rede ***INTERNET*** é muito frágil em termos de segurança, sendo necessário, portanto, estabelecer nossas próprias rotinas de segurança, que não podem ser menos seguras que o próprio sistema **UNIX**. Garantir a segurança do sistema é assegurar que nenhum usuário terá acesso a informações ou ações às quais não lhe seja permitido. Optamos por garantir essa segurança através de um sistema de senhas, no qual cada usuário tem a sua (similarmente ao sistema **UNIX**), e uma gama

UNICAMP
BIBLIOTECA CENTRAL

de rotinas de criptografia para manter o sigilo dos dados que trafegam na rede. Mais adiante esse esquema de segurança será melhor explicitado.

O sistema é modelado como ilustrado na Figura 11. Há uma interface conectada localmente através de **UNIX *pipes*** com um módulo cliente. Tal módulo é o responsável por abrir uma conexão via **RPC** com o servidor, e servir de ponte entre os dois módulos, executando as operações de criptografia necessárias. O servidor, por sua vez, é o responsável pela autenticação do usuário e pelo armazenamento e fornecimento dos dados. O armazenamento da base de dados pode ser feito através de um sistema gerenciador de banco de dados (**SGBD**), ou até mesmo de um sistema de arquivos. Outros dois módulos funcionam em cima da base de dados de um servidor. São eles: um sistema de correio eletrônico, que efetua as operações de aviso aos usuários do sistema, e um sistema de administração da base de dados, a ser operado somente pelo super-usuário daquele servidor. Cada servidor funciona de uma maneira independente. Uma questão que esteja sendo discutida em um servidor *X*, só pode ser acessada através desse servidor *X*. Até mesmo o cadastro dos usuários deve ser independente, ou seja, a identificação e a senha de acesso de um mesmo usuário pode variar de um servidor para o outro.

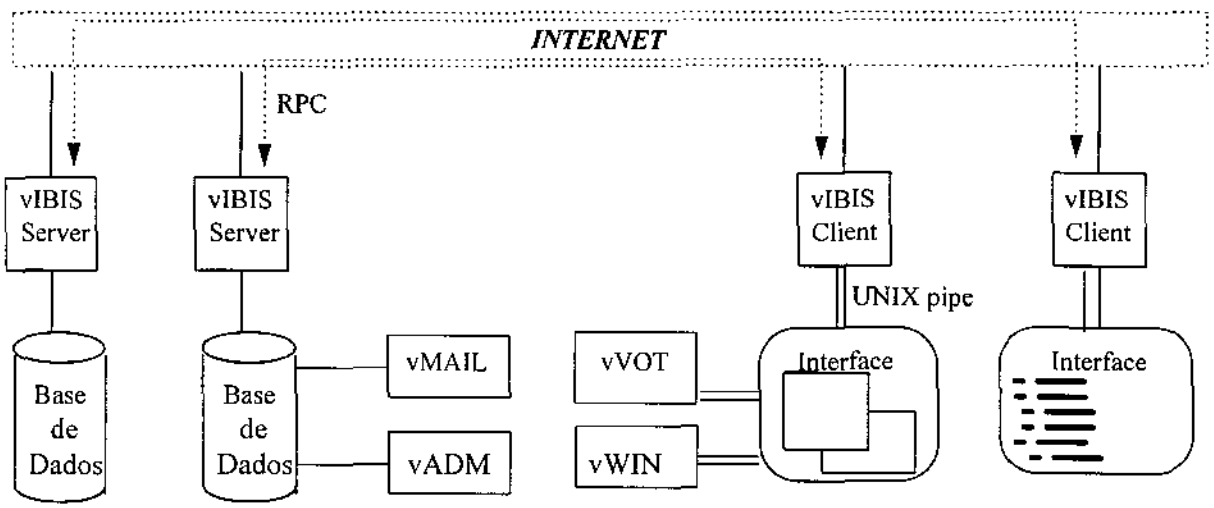


Figura 11 - Modelo de implementação de sistemas vIBIS

No lado do cliente, existem mais dois módulos que também possuem arquitetura aberta e podem ser implementados: o de votação e o de seleção dos vencedores. Esses módulos residem junto à interface, e um protocolo de comunicação (bem simples) entre eles também é especificado. A interface faz chamadas a esses programas que retornam o resultado: o voto do eleitor (no módulo de votação) ou a classificação das posições (no módulo de seleção dos vencedores). Dessa maneira,

qualquer pessoa pode criar seu próprio método de votação e incorporá-lo ao sistema através de uma simples configuração da interface. Outra grande vantagem dessas rotinas estarem do lado do cliente é que o método de seleção do vencedor não se torna uma caixa preta invocada remotamente, o que garante a qualquer usuário a possibilidade de realizar auditorias em cima dos resultados de uma eleição.

4.2 Conexão cliente/servidor

Em um sistema **vIBIS**, assume-se que sua operação se dá de forma distribuída, i.e., vários clientes em lugares diferentes fazem acesso a uma base de dados centralizada através de um servidor. Para que isso aconteça, deve haver um meio pelo qual as informações circulem. Dois processos podem se comunicar através de um *pipe* (uma conexão interprocessos tratada como um descritor de arquivo), porém precisam estar ativos em uma mesma máquina. Existe um mecanismo semelhante, que pode ser aberto entre dois processos em execução em máquinas distintas de uma rede **UNIX**, denominado *socket*. O uso de *sockets*, porém, por ser uma operação de nível muito baixo, é bastante complicado e, conseqüentemente, de difícil implementação.

Para abstrair o implementador de todas as operações em baixo nível necessárias em um sistema cliente/servidor (*sockets*, filas, estrutura de dados), existe o mecanismo denominado **RPC** (*Remote procedure call*) [SUN90A], que implementa, física e logicamente, o modelo cliente/servidor, englobando todas as operações de interação com a rede. Com RPC, um programa, que esteja sendo executado em uma máquina, pode fazer uma chamada de um procedimento em outra máquina. A chamada do procedimento remoto é exatamente igual à de um procedimento normal e, como tal, pode conter parâmetros e retornar valores. As rotinas embutidas em **RPC** se encarregam de fazer a transferência do parâmetro passado e de retornar o valor resultante da operação efetuada pelo procedimento. Existe também todo um suporte para definição dos mais variados tipos de dados e alocação de memória para armazená-los. Apesar de toda a discussão sobre **RPC** se referir a processos em máquinas diferentes, ele também funciona para processos diferentes na mesma máquina.

Cada procedimento de **RPC** é definido unicamente por um número de programa e um número de procedimento. O número de programa especifica um grupo de procedimentos relacionados, cada um com seu número de procedimento. Cada programa contém também um número de versão. Dessa forma, para um serviço

remoto, que pode ser chamado por qualquer usuário de qualquer lugar, existem o número de programa, versão e procedimento, que devem constar no manual desse serviço. Um usuário que deseje fazer um programa com chamada a esse procedimento, deve utilizar os números correspondentes. A cada programa que possa ser chamado remotamente, no momento da ativação, é atribuido um número de porta, que fica armazenado no *portmap* da máquina onde foi ativado. Todo o acesso ao programa se dá por essa porta, que pode ser alocada estática ou dinamicamente. Internamente, uma chamada de programa remoto faz primeiro uma chamada ao *portmap* do servidor (máquina onde o programa está ativo), único serviço de rede que tem porta dedicada, para saber a porta onde está alocado o programa, e só então faz a chamada ao programa através da porta especificada.

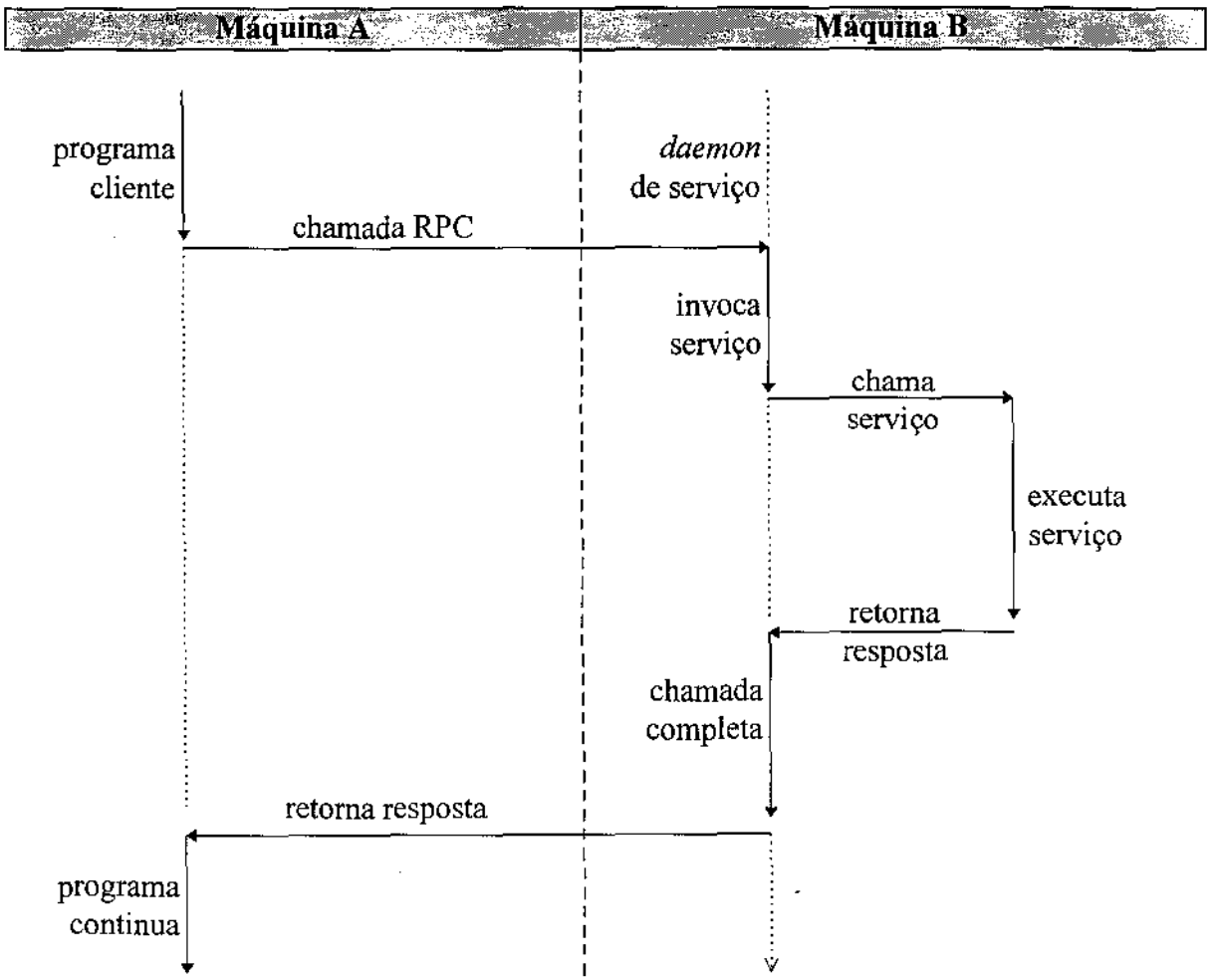


Figura 12 - Comunicação em rede com RPC

O funcionamento de uma chamada remota de serviço (procedimento) é descrito na Figura 12. O programa cliente gera a chamada, passa-a para um *daemon*

de serviços na máquina remota, que por sua vez, invoca o serviço, passando junto os parâmetros. Depois de executado o procedimento, a resposta é enviada de volta ao cliente, que continua sua execução. Em outras palavras, o programa cliente fica parado até a resposta retornar. Entretanto, existe também um parâmetro de *timeout* que pode ser alterado a critério do implementador. Se a resposta não chega no tempo programado, o controle volta ao programa cliente para que este possa tratar o erro de *timeout*. Se o procedimento remoto estiver sendo executado por um outro cliente, a chamada é colocada numa fila de espera e respondida pela ordem de solicitação (quando chegar sua vez).

Como forma de facilitar a geração de um programa com chamadas remotas, existe um pseudocompilador (rpcgen), que automaticamente gera todo o código necessário aos dois programas, tanto para o servidor como para o cliente. O programador constrói um arquivo com a especificação da chamada remota e executa o rpcgen em cima desse arquivo. O código gerado pode ser depurado ou modificado da maneira que se desejar.

Cliente e servidor **vIBIS** são dois programas que se comunicam via **RPC**; portanto, servidores deverão ter seus números de programa divulgados para que os clientes possam fazer acesso. A transmissão pode ser feita por um procedimento qualquer, onde os parâmetros contêm a mensagem do cliente e o retorno à mensagem do servidor. Em vez de utilizar um tipo bastante complexo para mandar as informações, e com o intuito de simplificar e facilitar a construção e implementação dos módulos, optamos por utilizar um único tipo de dado, ou seja, uma cadeia de caracteres de tamanho variável. O procedimento remoto fica então definido como:

```
char **vSTRING;
vSTRING send_message(vSTRING msg);
```

O arquivo a ser compilado pelo rpcgen é:

```
typedef string vSTRING<>;
program MESSAGEPROG
{
        version MESSAGEVERS
        {
                vSTRING sendmessage(vSTRING)=1;
        }=1;
}=0x20000098;
```

Mandar uma mensagem ao servidor nada mais é que fazer uma chamada à função *sendmessage* com a mensagem como parâmetro. O retorno da função é, então, a resposta do servidor.

4.2.1 Criptografia das mensagens

Como o sistema **vIBIS** se propõe a ser um sistema de apoio à tomada de decisões dentro de uma ou mais organizações, o conteúdo das informações das mensagens trocadas entre cliente e servidor pode ser de grande importância e sigilo para a organização. Além do mais, as senhas dos usuários que se conectam aos servidores também trafegam pela rede. Se alguém monitorar os sinais da rede, poderá ser capaz de descobrir e utilizar senhas de outros usuários. Por essa razão, as mensagens trocadas pelos módulos cliente e servidor, devem ser cifradas antes da remessa.

Senhas de usuários trafegam no sentido cliente-servidor. Não há nenhuma razão para que a senha trafegue no sentido contrário (servidor-cliente). Além disso, não deve haver nenhuma possibilidade de qualquer pessoa (inclusive o super-usuário) saber qual é a senha de outro usuário, por isso as senhas devem ser armazenadas (e até mesmo transportadas) cifradas, similarmente ao sistema de senhas do UNIX, por algoritmos do tipo unidirecional[23] [LUC86] (estabelecemos que será usado o mesmo algoritmo do **UNIX**, que inclusive está disponível através da função *crypt* em C/C++). Como estamos modelando um sistema aberto, onde qualquer pessoa pode implementar sua interface, com poder de se conectar a qualquer servidor, não

[23] Algoritmos de criptografia unidereccionais, geram mensagens que não podem ser decifradas.

podemos utilizar algoritmos de criptografia com chave única, pois qualquer pessoa poderia ter a chave de um determinado servidor e decifrar as mensagens a ele enviadas. Devemos, então, fazer uso de algoritmos com chave pública.

Algoritmos com chave pública são métodos de criptografia que se utilizam de duas chaves: uma pública, que qualquer pessoa pode conhecer, e outra secreta, que não pode ser divulgada. Uma mensagem cifrada com a chave pública só pode ser decifrada com a chave secreta, e vice-versa. Isso nos dá duas garantias: sigilo e autenticidade. No caso de um sistema cliente/servidor, onde as mensagens no lado do cliente são cifradas pela chave pública, e no lado do servidor decifradas pela chave secreta, garante que uma mensagem cifrada pelo cliente só possa ser lida pelo servidor (sigilo) e que as mensagens que puderam ser decifradas pelo cliente são realmente oriundas do servidor (autenticidade).

Escolhemos utilizar o algoritmo de chave pública conhecido por RSA, desenvolvido por Rivest, Shamir e Adelman [RSA78]. Esse algoritmo se baseia na escolha de dois números primos bastante grandes (da ordem de 10^{100}), os quais geram as chaves pública e secreta através de algumas operações (para maiores detalhes, consultar a bibliografia). A segurança do algoritmo está no fato de que as chaves públicas são compostas por números da ordem de 10^{200}, sendo possível se chegar às chaves secretas somente através da fatoração desses números. Um estudo utilizando o algoritmo de fatoração de Schroeppel (não publicado), o mais eficiente conhecido, mostrou que para tentar fatorar um número da ordem de 10^{200}, um computador levaria cerca de 10^9 anos, considerando que fosse gasto em cada operação matemática um tempo de 1μs [Luc86]. O **RSA** é considerado, pelos especialistas em criptografia, um bom algoritmo de chave pública.

Quando um servidor **vIBIS** é colocado em funcionamento, o desenvolvedor divulga, além das informações pertinentes à comunicação em rede (máquina, número de programa, etc.), o valor da chave pública utilizada. Dessa forma, os usuários que quisessem fazer acesso àquele servidor configurariam seus módulos clientes para cifrarem suas mensagens segundo a chave pública em questão. Isso garante o sigilo das mensagens emitidas pelo cliente, mas não garante o sigilo das mensagens-retorno. Para garantir o sigilo das mensagens-retorno, o cliente pode enviar, a cada mensagem, uma chave (pequena), gerada aleatoriamente em tempo de execução, a ser utilizada pelo servidor para enviar a resposta cifrada por essa chave. Nesse caso, um algoritmo de criptografia simples, por chave única, é utilizado. A própria variação da chave de criptografia já é um dispositivo de segurança das

mensagens que trafegam no sentido servidor-cliente. Nesse caso, não há nenhuma necessidade de autenticar o servidor.

Variáveis	Descrição
m	mensagem
Dkey	chave DES gerada pelo cliente
Pkey	chave pública do servidor
Skey	chave secreta do servidor

Funções
gera_key()
RSA()
DES_encrypt()
DES_decrypt()

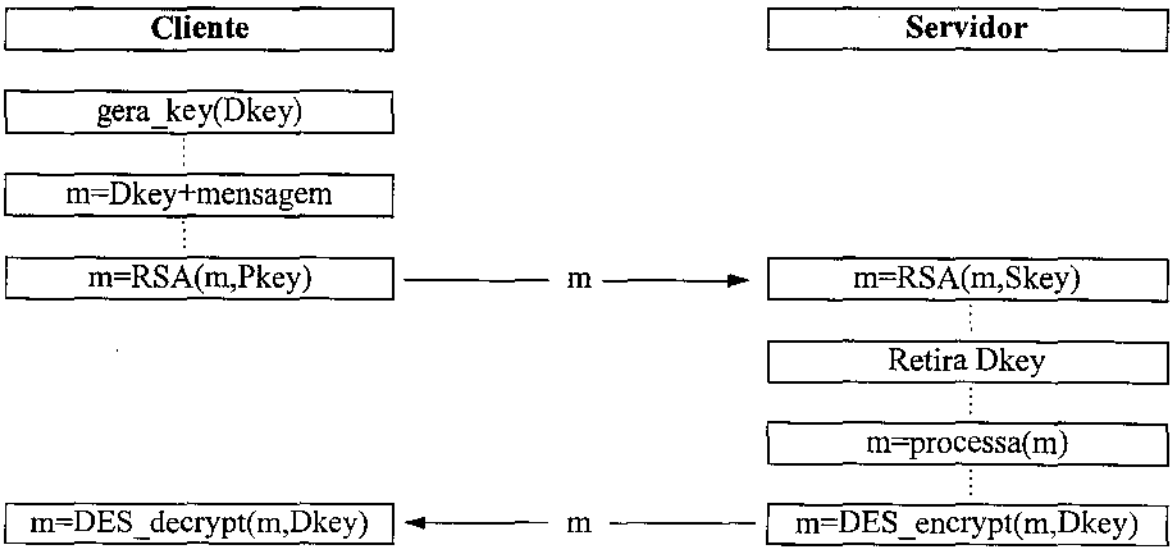


Figura 13 - Fluxo de criptografia entre cliente e servidor

Para as mensagens que trafegam no sentido servidor-cliente, optamos pelo uso do algoritmo simétrico **DES** (*Data Encryption Standard*) [KON81]. O algoritmo combina substituição (operação de troca dos caracteres) com permutação coordenada (um embaralhamento das letras dentro dos blocos), fazendo uso de uma chave de 64 bits (8 são de paridade). Esse algoritmo foi testado e aprovado pela **IBM**, **NSA** (*National Security Agency* - Agência de Segurança Nacional dos EUA) e adotado como padrão americano de ciframento de dados para aplicações não ligadas à segurança nacional pelo **NBS** (*National Bureau of Standards*). Para quebrar mensagens cifradas com o **DES**, seriam necessários 2^{56} ciframentos, cerca de 10^6 chips dias, a 1μs por ciframento. Isso dá bastante segurança ao nosso modelo, inclusive porque, além das chaves serem trocadas a cada mensagem, o que dificultaria o trabalho de um *hacker*, não há grandes problemas se alguém eventualmente conseguir ler as informações nesse sentido.

A representação da criptografia utilizada no sistema é apresentada na Figura 13.

4.2.2 Fluxo das mensagens

O fluxo das informações no sistema ocorre como descrito na Figura 14. O programa de interface ativa o módulo cliente, que abre uma conexão de **RPC** com o servidor. A interface passa, para o módulo cliente, qual servidor deve ser conectado e qual usuário, juntamente com sua senha, está pedindo a conexão (1). As informações pertinentes ao servidor (endereço IP, número de programa, chave de criptografia) devem constar de um arquivo qualquer, que o módulo cliente possa consultar para estabelecer a conexão. A primeira mensagem enviada ao servidor é o pedido de *login* do usuário (2), que retorna o número identificador desse usuário (3), ou zero, se o usuário não existe naquele servidor. Uma vez estabelecido o *login*, a interface faz pedidos ao servidor (4), mandando o comando, mais os argumentos, para o módulo cliente que serve de ponte entre os outros dois programas. A cada pedido é somado o *login* do usuário (número e senha), a chave de criptografia da resposta (5), e executada a rotina de criptografia utilizando a chave pública do servidor. O servidor checa se o *login* está correto e retorna ao cliente o resultado da operação efetuada, cifrada pela chave recebida (6). O cliente, por sua vez, repassa a resposta à interface já devidamente decifrada (7). O módulo cliente representa mais um nível de abstração para o implementador de interfaces, poupando-lhe o trabalho relativo à comunicação dos dados.

4.3 O servidor

Um servidor **vIBIS** deve prover o ambiente necessário ao armazenamento, transmissão, segurança e operações relativas à base de dados de um sistema **vIBIS**. Para o armazenamento, pode ser usado qualquer mecanismo capaz de gerenciar uma base de dados, como um SGBD, ou até mesmo um sistema de arquivos aliado a um conjunto de programas. A transmissão dos dados deve ser como a definida na seção anterior: via **RPC**. Porém, nada impede que surjam novas formas de servidores **vIBIS**, como por exemplo um servidor **HTML** compartilhando os dados com o servidor principal e dando acesso aos usuários via **WWW** (*World Wide Web*). Para manter a segurança, o servidor deve: (a) impedir ações de usuários sobre dados que não lhes sejam permitidos; (b) dar acesso aos usuários somente através de senhas; (c) não conter formas de recuperação, nem mesmo por parte de super-usuários, de

senhas, votos secretos e/ou autores de contribuições anônimas; (d) assegurar a consistência dos dados por ele armazenados.

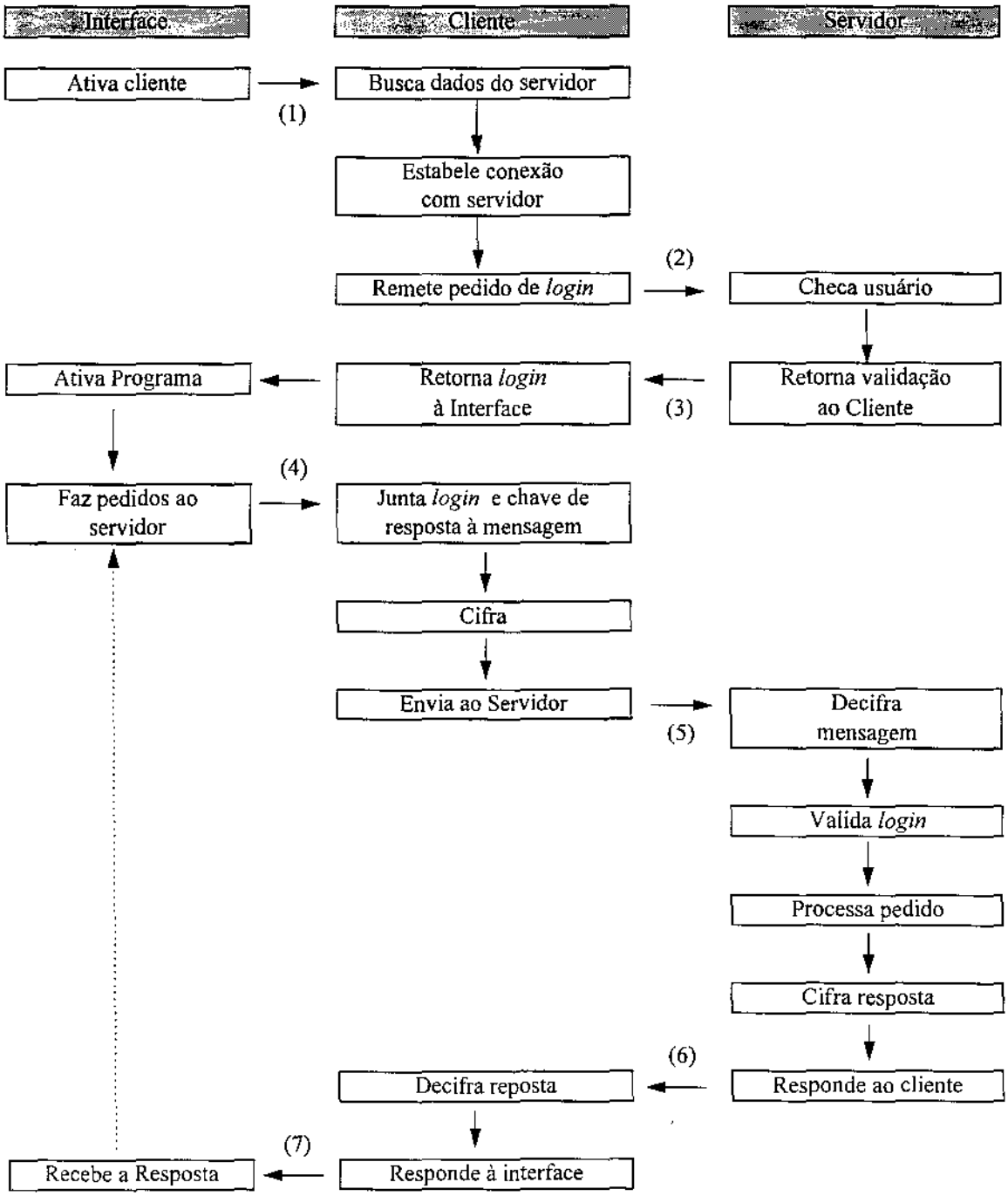


Figura 14 - Fluxo de mensagens

Finalmente, é também necessário que o servidor tenha implementados todos os comandos especificados no protocolo **vIBIS** de comunicação. Novos

comandos podem também ser implementados, devendo então serem utilizadas as faixas reservadas para tal fim (ver Anexo I). Na concepção básica de sistemas **vIBIS**, um servidor deve ser alcançável através da ***INTERNET*** por qualquer máquina, porém não descartamos a possibilidade de servidores "secretos", com acesso somente através de determinadas máquinas, por exemplo, servidores dedicados a discussões internas de uma organização.

4.3.1 Conexão com o cliente

Como visto na seção anterior, um servidor **vIBIS** deve implementar um procedimento, executado via **RPC**, segundo a definição apresentada anteriormente. Esse procedimento está em um programa executado como um *daemon*, que espera chamadas dos clientes. As chamadas são automaticamente organizadas em fila e o programa servidor atende uma de cada vez. Essa tarefa é efetuada por funções de **RPC** de nível mais baixo e é transparente para o implementador.

Uma padronização no número de programa dos servidores **vIBIS** facilita bastante a tarefa dos usuários de se conectarem aos servidores. No protótipo que desenvolvemos, utilizamos o número 0x20000098, o qual sugerimos que seja utilizado por implementadores de outros servidores, observando, porém, se esse número não está sendo utilizado por algum outro aplicativo no mesmo *site*.

Para se conectar a um servidor, o cliente deve estar de posse, além do número de programa, da chave pública que serve para cifrar as mensagens a ele enviado. Cada servidor deve ter as suas chaves pública e secreta a serem utilizadas para cifrar e decifrar as mensagens com o algoritmo **RSA** [RSA78], a fim de garantir que as mensagens que trafegam na rede não sejam violadas.

O procedimento do servidor recebe uma mensagem cifrada, segundo a chave pública do servidor, a ser decifrada com a chave secreta por uma rotina qualquer que implemente o **RSA**. Fica livre a escolha da rotina a ser utilizada, podendo o implementador inclusive fazer opção por um programa externo, como, por exemplo, o **PGP** (*Pretty Good Privacy*), que pode receber a mensagem como parâmetro e retorná-la decifrada. Programas como esse são interessantes, uma vez que já são otimizados para obter a melhor desempenho.

A mensagem oriunda do cliente, depois de decifrada, deve ser analisada sintática e semanticamente. Entretanto, pode acontecer da mensagem ter sido cifrada erroneamente, por exemplo, com uma outra chave que não a do servidor, ou até mesmo por erro da rotina de criptografia no lado do cliente. Nesse caso, o servidor não tem nada a fazer a não ser retornar uma mensagem de erro. Note-se que, nesse

caso, o retorno não pode ser cifrado, pois o servidor não terá tomado conhecimento da chave sorteada pelo cliente. Por essa razão, o primeiro caracter das mensagens retorno, que são o resultado da operação (ver "Código de resultados" no Anexo I), nunca devem ser cifrados. As mensagens que trafegam no sentido servidor-cliente só são cifradas do segundo caracter em diante.

Cada chamada por **RPC** é tratada, pelo servidor, como uma chamada atômica, ou seja, o início e o fim de conexão acontece para cada chamada, e não para toda a execução do programa cliente. Não há uma maneira de o servidor tomar conhecimento de que o cliente terminou sua execução. Poderíamos implementar uma espécie de "sessão" entre cliente e servidor, porém, nada garantiria que o programa cliente encerrasse a conexão. Poderia acontecer o caso de o programa ser abortado durante a execução do cliente, sem que o fechamento da seção fosse enviado ao servidor. Por essa razão, segundo o protocolo de comunicação definido, toda mensagem no sentido cliente-servidor contem número de identificação e senha do usuário **vIBIS** e o servidor deve autenticar o usuário sempre. Para isso, o servidor pode manter uma tabela cache na memória com os últimos usuários que fizeram acesso. Caso o usuário não conste na tabela, a base de dados deve ser consultada.

4.3.2 O interpretador

O servidor deve conter um *parser* para interpretar as mensagens recebidas do cliente. Essa é a parte mais importante do servidor, e faz a principal tarefa, que é atender aos pedidos do cliente. O interpretador deve, após identificar o comando, fazer a checagem dos direitos de acesso do usuário em questão, segundo os critérios do modelo **vIBIS**. Após essa checagem, deve ser feita a análise semântica do comando, a execução e a preparação da resposta a ser retornada ao cliente. Em qualquer uma dessas fases, pode ser encontrado um erro, que deve, então, ser transformado em mensagem retorno a ser enviada ao cliente. As ações que devem estar implementadas no interpretador estão descritas na seção de descrição dos comandos do protocolo (Anexo I).

De posse da resposta, a mensagem é cifrada, através de uma rotina que implementa o algoritmo **DES**, com a chave da resposta enviada pelo cliente, a fim de evitar que alguém monitorando a rede possa ler a resposta. Assim como o **RSA**, existem alguns programas que já implementam o **DES** de maneira bastante eficiente, que podem ser utilizados na criptografia da resposta.

4.3.3 Servidores paralelos

Dependendo do número de usuários e do grau de utilização do sistema por parte desses usuários, a existência de somente um servidor pode ser insuficiente para atender a tantos pedidos. Essa insuficiência gera uma quantidade muito grande de *timeouts* no lado dos clientes, além de degradar bastante a velocidade do sistema como um todo. A solução para esses casos seria a implementação de vários programas que trabalhem paralelamente em cima da base de dados. Esses programas, logicamente, têm de ser identificados com números de programas diferentes. Um programa central, igualmente com uma função **RPC**, seria chamado pelos clientes logo que entrassem em execução. O programa central analisaria o uso dos diversos servidores e retornaria ao cliente o número de programa do servidor que está menos ocupado. O cliente, então, passaria à execução normal, chamando o servidor escolhido.

Vale lembrar que os servidores devem se preocupar com a consistência dos dados que estão sendo compartilhados. Como a maioria das ações dos usuários detém-se à consulta da base de dados, operações de *lock* são menos freqüentes, o que torna o sistema mais rápido. Se todas as operações exigissem o travamento dos dados, a existência de vários servidores teria pouca valia.

4.3.4 Sistema automático de *e-mail*

Mencionamos, no Capítulo 3, que uma das vantagens de um sistema automático de discussão e votação é que ele pode, automaticamente, lembrar aos usuários sobre os acontecimentos. Os usuários podem ser avisados sempre que uma nova decisão, à qual eles terão acesso, for aberta, ou sobre períodos de votação que acabaram de se iniciar, ou sobre resultados da votação disponíveis, etc. Reservamos um campo de tamanho de um *byte* na tabela de usuários (Ver Anexo II), no qual cada *bit* corresponde à escolha do usuário sobre o recebimento de mensagens de aviso (1 para "sim" e 0 para "não"). Essas opções são:

- sempre que um novo grupo de discussão (que seja acessível) for criado;
- sempre que for incluído em algum grupo;
- sempre que uma nova questão for incluída;
- sempre que atingir o "*deadline*" para a inclusão de novas contribuições (posições, argumentos, *etc.*);
- sempre que se iniciar um período de votação;

- sempre que votar (essa opção é uma forma de o usuário saber se sua identidade está sendo usada indevidamente);
- sempre que alterar esses parâmetros de *mail* (a finalidade dessa opção é a mesma do item anterior; o intruso inteligente desligaria o item anterior antes de votar, o que geraria uma mensagem de alteração de parâmetros).

Essas sete opções devem estar disponíveis em todos os servidores **vIBIS** e o usuário deve poder configurá-las a seu critério.

O sistema de *e-mail* automático deve assegurar que as mensagens sejam realmente enviadas, podendo, inclusive, incorporar rotinas de tratamento automático de mensagens que não cheguem a seu destino (*undelivered mail*). De qualquer maneira, o super-usuário deve tomar conhecimento dessas mensagens a fim de efetuar as devidas ações. Uma importante tarefa para esse sistema é relativo à inclusão de novos usuários ao servidor. Não há como garantir que um usuário qualquer, solicitando sua inclusão ao servidor, seja realmente quem diz que é. Qualquer pessoa pode, por exemplo, pedir para incluir o usuário *fulano@algum.lugar*. Como o acesso ao servidor só se dá através da senha do usuário, o servidor sorteia uma senha, aleatoriamente, no momento da inclusão e a remete através de *e-mail* para o endereço alegado (*fulano@algum.lugar*). Assim o usuário, se existir, toma conhecimento de que alguém solicitou sua inclusão. As únicas formas de ter acesso à essa mensagem que contém a senha é, ou monitorando as mensagens que passam entre a máquina do servidor para a máquina do usuário, ou sendo *root* no *site* do usuário. Supõe-se que um *root* dentro de um domínio é sempre confiável, pois dele depende toda a segurança do sistema **UNIX** local. Para alguém monitorar a rede entre os dois *sites*, é preciso que essa pessoa esteja em um dos roteadores entre as duas máquinas e que tenha privilégios de *root*. Além disso, essa não é uma tarefa fácil; dificilmente alguém conseguiria efetuar tal operação. Essa discussão, entretanto, leva-nos a uma discussão bem mais abrangente, ou seja, a segurança dentro da própria ***INTERNET***, o que foge completamente ao escopo desta tese.

Um usuário pode ser incluído no servidor de duas formas: pelo gerente de um grupo de discussão, ou por pedido do próprio usuário. As duas formas geram uma senha automática a ser divulgada ao usuário pelo sistema de *e-mail* através de mensagens do tipo: "Você pediu sua inclusão nesse servidor. Sua senha é 'passwd'. Mude-a o mais breve possível.".

O servidor deve controlar que ações sobre a base de dados causam a geração de mensagens e a quem essas mensagens devem ser enviadas. Por exemplo, a inclusão de uma questão para ser decidida por um grupo qualquer deve gerar mensagens de *e-mail* para todos os integrantes desse grupo que estejam configurados para receber mensagens de inclusão de questões.

4.3.5 Super-usuário

É necessário a um sistema que faça uso de uma base de dados, a existência de um usuário responsável pela manutenção e consistência dos dados envolvidos. No sistema **vIBIS** existe a figura do super-usuário do servidor. Cada servidor tem uma pessoa responsável por várias tarefas que não podem ser consideradas operações **vIBIS**, como rotinas de *back-up*. Além disso, existem operações que não podem ser permitidas a nenhum usuário normal, nem mesmo a gerentes de grupos ou questões. Suponha o caso de um usuário que tenha pedido sua inclusão e usou um endereço de *e-mail* errado. A mensagem que o servidor remete ao usuário com sua senha, não chega a seu destino e retorna como *"undelivered mail"*. Nesse caso, a exclusão do usuário erroneamente cadastrado não pode ser realizada por nenhum outro usuário.

No caso de algum usuário esquecer sua senha, ele remete mensagem ao super-usuário do servidor pedindo uma nova senha. O super-usuário dispara um processo de geração aleatória de senha que, como na inclusão, remete a nova senha por *e-mail* ao usuário. Todas as ações relativas a uma questão ou a um grupo de discussão podem ser executadas por seus gerentes, porém qualquer outra ação fora desses dois domínios tem que ser tomada pelo super-usuário.

4.4 O módulo cliente

O módulo cliente é responsável por repassar as mensagens enviadas da interface para o servidor e vice-versa. Esse módulo foi criado para aumentar o nível de abstração do implementador da interface, embora interface e cliente possam estar agregados em um só módulo.

Este módulo é executado paralelamente a partir da interface, que abre dois *pipes* para que os dois se comuniquem entre si. Quando a interface solicita uma chamada ao servidor, remete essa chamada ao cliente, pelo *pipe* de escrita, que faz o **RPC** correspondente para o servidor. Durante a chamada ao servidor, o módulo cliente fica bloqueado, esperando a resposta do servidor, porém a interface continua sua execução normalmente e pode atender a qualquer evento de *mouse*, teclado, etc.

Quando a resposta do servidor chega, o módulo cliente escreve essa resposta no *pipe* de leitura da interface. Um notificador na interface gera um evento, avisando que há alguma coisa a ser lida no *pipe*. Se durante o bloqueio do cliente, a interface fizer outro pedido, esse fica no *pipe* de escrita até que o bloqueio termine e o cliente esteja livre para o processar. O módulo cliente permanece em um laço aguardando algum pedido da interface no *pipe* de escrita. Quando chega algum pedido, remete ao servidor, devidamente identificado e cifrado, e, após chegar a resposta, coloca-a no *pipe* de leitura e retorna ao laço à espera de novas mensagens oriundas da interface.

No caso do protótipo implementado, na execução do cliente, a interface remete, como parâmetro, o nome do *site* onde se encontra o servidor ao qual o usuário deseja se conectar. Obtendo sucesso na conexão, é remetido o pedido de *login*, que não passa de uma conferência da senha e um pedido do número do usuário naquele servidor. Se a senha está correta, o servidor retorna o número desse usuário, repassado à interface, informando que o *login* está correto. A partir disso, o módulo cliente, conhecendo o número e a senha do usuário, passa a remeter esses dados junto com cada mensagem enviada ao servidor, devido à atomicidade das mensagens, descrita na seção anterior.

Da mesma forma que o módulo servidor, o cliente também faz uso do **PGP**, um programa externo de criptografia que implementa o algoritmo de chave pública **RSA**, para cifrar as mensagens enviadas ao servidor. No protótipo desenvolvido, a chave pública e o número de programa dos servidores conhecidos estão armazenados em um arquivo, que o módulo cliente consulta antes da conexão.

4.5 A interface

Dos módulos que compõem o lado do cliente no sistema **vIBIS**, o programa que faz a interface entre o sistema e o usuário pode ser considerado o principal. Através da interface, o usuário dispara as ações que compõem a discussão e votação em cima das decisões a serem tomadas. O programa de interface deve suportar o modelo **vIBIS** nos seus dois aspectos: discussão e votação. A parte relativa à discussão, como mencionamos no capítulo anterior, poderia ser adaptada a qualquer programa baseado no modelo **IBIS** de discussão, desde que esse atendesse aos requisitos relativos à arquitetura cliente/servidor do modelo **vIBIS**, e englobasse as pequenas alterações ao modelo **IBIS** original, o que já foi incorporado em alguns deles (**gIBIS**, **CM/1**, etc.). Sistemas **IBIS** não provêm formas de determinar se uma discussão em uma determinada questão foi encerrada ou resolvida. No modelo **vIBIS**,

uma questão termina no momento em que foi escolhida uma posição, através de votação, como sendo a solução para a questão levantada. A interface deve prover meios para que esse fato fique bem explícito para o usuário.

O programa de interface pode ser implementado utilizando qualquer forma de apresentação, seja ela em forma de janelas ou até mesmo em linhas de comando. Interfaces podem ser criadas para plataformas como **Emacs**, **Netscape**, **Mosaic**, ou outras. Para interfaces que se utilizem de um ambiente gráfico, pode ser usado qualquer *toolkit* como **Xview**, **SUIT**, **Motif**, **Sunview**, etc. O único requisito é que o programa atenda às especificações do modelo. Durante nosso curso de mestrado, desenvolvemos protótipos de interfaces gráficas para **Xview** [HEL93] e **SUIT** [CPP92]. Também foi implementada uma interface de linha de comando que, entre outras coisas, exigiu a criação de uma linguagem própria, o que dificulta bastante sua utilização. Esse projeto, assim como a interface **SUIT**, acabou sendo abortado antes de chegar a um produto final.

Esse módulo pode incorporar todos os outros módulos no lado do cliente, i.e., não há estrita necessidade do módulo cliente nem dos módulos de votação e seleção de vencedores separados. A própria interface pode efetuar as operações que esses programas executam, porém o conceito de livre implementabilidade de novos métodos de seleção do vencedor fica reduzido a implementadores de interfaces. Por essa razão, acreditamos que deve haver pelo menos uma maneira de comunicação implementada entre a interface e os programas relativos à eleição que por ventura algum usuário deseje implementar. No protótipo que desenvolvemos, a interface já incorpora alguns métodos de seleção do vencedor, porém existe uma porta de comunicação para que sejam conectados outros programas que implementem novos métodos.

4.5.1 O funcionamento

Logo no início da execução, a interface deve conhecer que servidor será contactado e a que usuário será feito o pedido de *login*. Tal pedido de *login* deve acompanhar endereço completo do usuário (nome e domínio) e sua senha junto ao servidor. Esses dados podem ser obtidos através da linha de comando do programa, de uma digitação em tempo de execução ou até mesmo de um arquivo de configuração, que pode estar localizado no diretório do usuário. O ideal é que a interface incorpore as três opções apresentadas. Logicamente, se a senha também fizer parte de um arquivo de configuração, ela deve estar criptografada e protegida contra leitura por outros usuários. A senha deve ser entrada em tempo de execução,

para dificultar os ataques de *hackers* que, eventualmente, possam vir a tentar se conectar ao sistema através da conta de outra pessoa. Entretanto, para o nome do usuário e dados do servidor mais freqüentemente utilizados, o armazenamento em um arquivo de configuração torna-se bastante vantajoso, pois evita a tarefa cansativa de digitação toda vez que o programa é executado.

De posse dos dados do usuário é realizado, então, o pedido de *login*, que retornará o número do usuário, caso esse exista e a senha esteja correta. A partir de então, a interface e o módulo cliente estão de posse dos dados necessários para fazer as chamadas do procedimento remoto do servidor contendo os comandos especificados no protocolo **vIBIS**.

Cada comando **vIBIS** é tratado atomicamente no servidor e não depende de nenhuma seqüência para ser executado. Na interface, entretanto, há necessidade de uma certa ordem nas operações. Por exemplo, na primeira tela, o normal é um "Menu" com as questões que o usuário tem acesso. A interface pede, então, uma lista dessas questões e, em seguida, logo que o usuário escolher a questão a ser visualizada, é enviado o pedido dos dados constantes nesta questão. Nada impede de alguém construir uma interface que receba os próprios comandos **vIBIS** da maneira como são especificados no protocolo. Nesse caso, o primeiro comando do usuário pode ser para incluir um argumento *X* em uma questão qualquer. Os comandos são atômicos e completos. Por exemplo, se algum comando exige uma senha, a interface deve se antecipar e pedir essa senha ao usuário antes de remeter o pedido ao servidor.

No Anexo IV descrevemos brevemente o protótipo final da interface que faz parte do protótipo implementado, que deve servir de exemplo para novas implementações de interfaces **vIBIS**. Esse programa passou por várias etapas, até a versão atual. Versões preliminares estão descritas em [CW94A] e [CW94B].

4.6 Os módulos de métodos de votação

Como foram descritos no Capítulo 2, métodos de votação, que são formados pela união de uma forma de votação com um método de seleção do vencedor, podem possuir uma infinidade de variações. Neste trabalho não advogamos a favor de nenhum método de votação e, por essa razão, não podemos impedir aos usuários do sistema que façam qualquer escolha dentro do universo de métodos concebíveis. Se o usuário deseja implementar seu próprio método, o sistema deve estar apto a incorporá-lo. Poucas pessoas podem ter acesso ao lado do servidor e a implementação de um novo método depende, inclusive, de permissão por parte do

administrador do servidor. Por isso, modelamos o método de votação e seleção do vencedor no lado do cliente. As rotinas para suportar as ações de voto e sua contagem podem já estar incorporadas ao código da interface, mas novos métodos que venham a surgir podem ser incorporados como programas à parte.

Uma rotina de votação deve, dado um conjunto de posições que podem receber votos, direcionar o usuário corretamente sobre essas posições e retornar à interface o voto do usuário no formato que especificaremos mais adiante. Da mesma forma, devem funcionar as rotinas de seleção de vencedores, ou seja, dado um conjunto de votos, a classificação das posições deve ser retornada. As interfaces devem ser configuráveis para fazer chamadas a esses programas externos.

Segundo a descrição das tabelas da base de dados das decisões (apresentada no Anexo II), a cada turno da eleição está associado um campo para a forma de votação, outro para o método de seleção do vencedor e outro para o método de desempate que será utilizado. Esses campos são definidos como do tipo Char(10), i.e., uma *string* de 10 caracteres. No servidor, não é feita nenhuma consistência dos dados contidos nessas *strings*. O gerenciamento e a consistência desses dados são feitos pela própria interface. Esperamos que esses dez caracteres sejam suficientes para a definição dos métodos que eventualmente venham a ser desenvolvidos. No protótipo, utilizamos o primeiro *byte* para a definição do método e os restantes para parâmetros desse método que eventualmente tenham que ser definidos. Por exemplo, para o método *Standard-a×b*, definimos a letra "A" para identificá-lo, concatenado com *a* e *b*, representados cada um em dois caracteres (A definição para *Standard-2×1* seria "A0201"). É interessante manter uma padronizaçao no formato da identificação dos métodos, para que dados processados em uma interface também possam ser processados em outra. Portanto, cada vez que surgir um novo método, o super-usuário do servidor deve ser contactado para a criação da padronização do novo método. O implementador da interface deve estar atento para os casos que possam fazer referência a métodos não implementados e emitir uma mensagem ao usuário informando que tal método não está implementado localmente. Os métodos por nós desenvolvidos já contam com essa padronização e são apresentados no Anexo III.

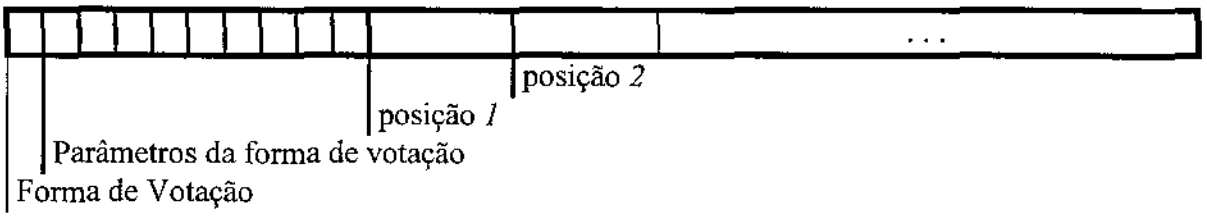
O programa da interface deve consultar um arquivo de configuração no qual estão especificados os métodos de votação disponíveis na máquina local. No arquivo devem constar todas as informações relativas às formas de votação e métodos de seleção dos vencedores, incluindo um campo que indique quais métodos de seleção de vencedores estão disponíveis para cada forma de votação. Dados como número, nome, tipo e valores limite dos parâmetros também são necessários, pois

serão utilizados nas rotinas de configuração dos turnos de votação que a interface implementa. Além disso, deve constar o *path* completo dos programas que aplicam os métodos, caso esses sejam implementados externamente.

A comunicação entre a interface e o método pode ser feita através de **UNIX** *pipes*, similarmente à comunicação com o módulo cliente. Nos programas de seleção de vencedores, é interessante que seja retornado, além do resultado, um texto explicativo sobre os cálculos efetuados para facilitar a auditoria do método por parte dos usuários.

4.6.1 Comunicação interface-rotina de voto

A interface deve remeter ao programa, que direcionará o voto do usuário, os dados constantes no campo que determina a forma a ser utilizada, para que o programa possa configurar-se pelos parâmetros aí existentes. Devem ser remetidas, também, as posições que poderão receber votos, segundo o parâmetro de tipo de nó (votos em nível ou nas folhas), que são definidos para cada turno da eleição. Essas informações devem seguir um formato padronizado da seguinte forma:



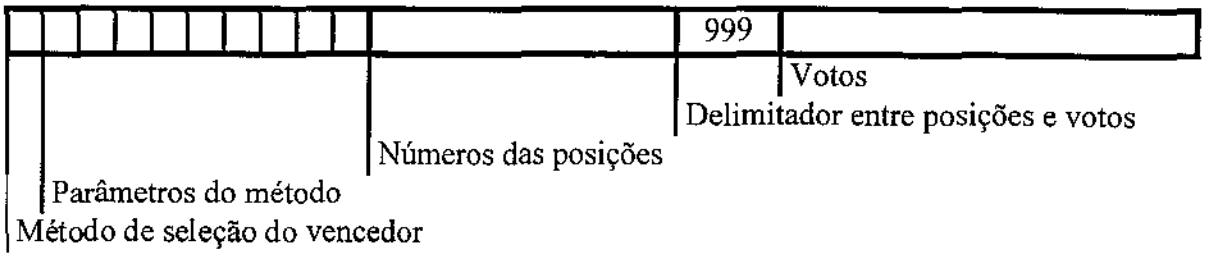
Cada posição contém seu número (3 caracteres) e sua descrição (25 caracteres).

O retorno do programa deve conter as informações necessárias para o programa que fará a classificação das posições, isto é, o voto do usuário. O formato do retorno contém o número das posições votadas (ou vetadas) mais um valor que deve ocupar 5 caracteres. Para o caso de votos por notas, o programa retorna as posições com a nota atribuída a cada uma delas. No método de pluralidade, o programa só precisa retornar a posição votada, seguida de 5 caracteres vazios, no campo relativo aos valores (os valores não são necessários, mas devem ocupar o espaço reservado a eles). O programa de classificação deve saber o que significam os valores associados a cada posição.

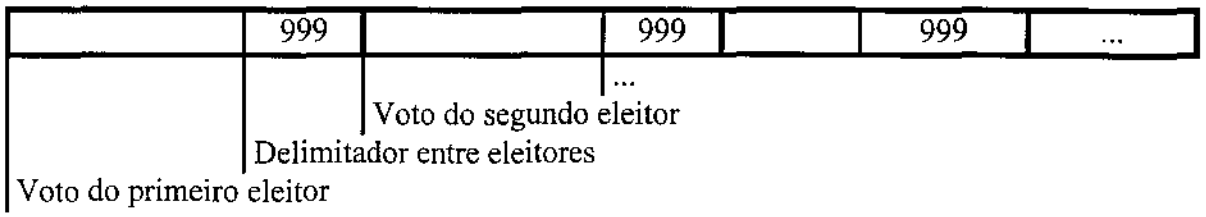
Depois de receber o voto do usuário, a interface deve repassá-lo ao servidor para que seja armazenado.

4.6.2 Comunicação interface-rotina de classificação das posições

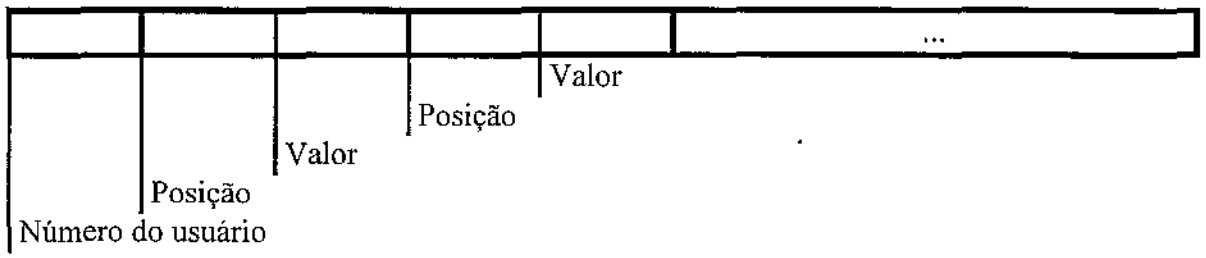
A interface deve fazer o pedido dos votos ao servidor, que apresentam o formato descrito no Anexo II, e repassá-los à rotina de classificação juntamente com os parâmetros do método de cálculo e os números das posições. O formato dessa mensagem é o seguinte:



Os números das posições são concatenados um a um. Os votos obedecem ao seguinte formato:



O voto de cada eleitor se configura como:



O número do eleitor (de 5 dígitos) é omitido (5 dígitos vazios) caso o voto seja secreto. Cada posição votada por esse eleitor apresenta seu número (3 dígitos) e o valor associado pelo programa de votação (5 dígitos). Quando, ao invés de um

número de posição, for encontrado o número 999, é sinal que os próximos votos pertencem a outro eleitor.

O programa externo deve, após receber os dados, compilá-los, segundo seu critério, gerar um relatório em arquivo (que pode ser apresentado ao usuário pela interface) e retornar o resultado contendo cada posição e sua colocação, segundo o método utilizado. Caso ocorram empates, as posições empatadas apresentam a mesma classificação. A interface analisa os dados e, se preciso, aciona a rotina de desempate para as posições que ficaram empatadas. A classificação final é, então, apresentada ao usuário, juntamente com os *reports* dos programas externos. Se existe alguma inconsistência, seja nos parâmetros do método ou nos próprios votos, o programa deve especificar o erro no relatório.

Eventualmente, esse resultado precisa ser passado ao servidor para que esse efetue o fechamento do turno de votação e só considere, nas votações seguintes, as posições que se classificaram. Essa tarefa só pode ser efetuada pelo gerente da questão, para evitar que qualquer pessoa mande o resultado da maneira que desejar. Mas, e se o gerente for desonesto e utilizar uma classificação, que não seja a corretamente calculada pelo método? Nesse caso, existe sempre a possibilidade de os usuários pedirem o cálculo do resultado da eleição para aquele turno. Se o resultado não coincidir com os que estão gravados no servidor, é sinal que o gerente não utilizou o método correto.

A escolha de um método de votação por parte de um grupo de discussão está diretamente ligada à confiabilidade que os membros do grupo têm no programa que implementa tal método. Não se deve aceitar utilizar um método sem antes fazer uma espécie de auditoria no programa. Note-se que os dois módulos relacionados ao método devem estar disponíveis nas máquinas de todos os usuários que farão uso deles. Uma opção para métodos de cálculo dos vencedores é a implementação de módulos que façam uma chamada remota, via **RPC**, de um processo que está em outra localidade da ***INTERNET***. Isso é perfeitamente possível, desde que esse processo remoto retorne as mesmas informações especificadas nesta seção. O módulo local, nesse caso, serve de ponte entre a interface e o programa de classificação. Dessa forma, podem, eventualmente, surgir *sites* servidores de métodos de votação.

4.7 Conclusão

Esse capítulo apresentou o modelo de implementação para sistemas **vIBIS**. Sistemas **vIBIS** trazem, como características, distribuição, arquitetura aberta

e modularidade. A distribuição do sistema foi incorporada através de um modelo cliente/servidor, que se utiliza de chamadas **RPC** para a comunicação entre os dois lados. O sigilo das informações que trafegam na rede é garantido por um conjunto de procedimentos e programas de criptografia, que cifram e decifram as mensagens trocadas entre clientes e servidores segundo dois algoritmos clássicos em criptografia: **DES** e **RSA**. Cada lado (cliente ou servidor) é de arquitetura completamente aberta e pode ser implementado por qualquer pessoa. Além dessa arquitetura aberta, modelamos também um lado cliente modularizado ao qual podem ser incorporadas novas rotinas de votação e/ou seleção de vencedores. Por fim, definimos um protocolo de comunicação entre a interface e esses módulos agregados.

Conclusões

Nesta dissertação apresentamos um modelo de discussão e votação denominado **vIBIS**, cuja estrutura de votação foi baseada no modelo **IBIS** com pequenas alterações para suportar, além de discussão, um domínio de votação. O modelo **IBIS** original não apresentava nenhum mecanismo para a resolução das questões ora em discussão. Criamos esse mecanismo através de processos de votação. Resolver uma questão é, dado o conjunto de objetos **IBIS** (questões, posições *e* argumentos), escolher uma posição, i.e. uma das soluções apontadas por um dos participantes da discussão, através de um processo qualquer. Nesse caso escolhemos o processo de votação.

Fizemos algumas extensões no modelo **IBIS** original, sendo algumas delas já adotadas em sistemas **IBIS**-*based* conhecidos na literatura (**gIBIS**, **CM/1**). Essas extensões foram: criação de níveis hierárquicos para posições (generalização/especialização de posições); criação de um tipo de nó que pode se ligar a qualquer dos 3 tipos de objetos (questões, posições ou argumentos) para abrigar outros tipos de contribuições à discussão; e, finalmente, a redefinição de argumentos como comentários sobre as posições, o que permite comentários indiferentes além dos originais de objeção ou suporte.

No Capítulo 3, fizemos uma exposição sobre vários sistemas de votação, alguns deles considerados clássicos pela literatura. Criamos uma modelagem para esses métodos, subdividindo-os em "Formas de Votação" e "Métodos de Seleção de Vencedores". Essa modelagem visa facilitar a implementação de tais métodos.

Uma votação pode se dar de várias formas: o eleitor pode escolher um número x de candidatos que gosta e um número y dos que não gosta, pode dispor os candidatos na forma de um *ranking*, indicando assim suá ordem de preferência, pode dar notas a cada candidato, etc. Dependendo da forma de votação utilizada, podemos aplicar diversas funções nos votos dos eleitores para determinar a classificação dos candidatos. Por exemplo, se a forma de votação utilizada foi através de nota, poderíamos determinar que o vencedor seria o candidato que obtivesse a maior média de notas. Essas funções determinantes de uma ordem de classificação dos candidatos foram, por nós, denominadas "Métodos de Seleção dos Vencedores". O conjunto de

métodos de seleção de vencedores para cada forma de votação pode ser praticamente infinito, i.e., podemos determinar qualquer critério a ser utilizado.

Além do modelo **vIBIS**, que organiza e estrutura uma decisão, especificamos também um modelo de implementação para sistemas **vIBIS**. Tal modelo se baseia na arquitetura cliente/servidor e tem como principais características: distribuição, arquitetura aberta e modularidade. Sistemas **vIBIS** devem ser implementados em módulos, no lado cliente ou servidor, e devem conter portas de comunicação com outros processos que porventura venham a ser desenvolvidos. Por exemplo, uma nova interface pode ser desenvolvida e deve poder se comunicar com qualquer servidor **vIBIS** previamente existente; um novo método de votação pode ser implementado e o sistema deve estar preparado para incorporá-lo sem que haja, por exemplo, necessidade de mudanças nas interfaces para comportar essa nova forma de votação. Para manter essa abertura e implementabilidade de qualquer módulo do sistema, especificamos protocolos de comunicação entre clientes e servidores (Anexo I - Protocolo **vIBIS** de comunicação) e entre interfaces e rotinas de votação.

A utilização de um sistema automático para deliberações pode trazer muitos benefícios para o grupo de pessoas que o utiliza:

- As pessoas não precisam, necessariamente, estar em um mesmo local no mesmo momento, para discutir sobre as questões;
- Os usuários podem ter mais tempo para preparar seus argumentos e pensar sobre a opinião dos outros;
- Opiniões anônimas podem ser permitidas, melhorando o aspecto democrático da discussão: ninguém deixa de expor suas idéias por medo de retaliações;
- As atenções são concentradas mais nas questões e menos nas pessoas; é mais difícil pessoas de personalidade mais forte dominarem a discussão;
- Contribuições fora dos assuntos em pauta são desencorajados pela própria estruturação da discussão, i.e., o usuário só pode inserir objetos **IBIS**.
- As discussões podem ficar amarzenadas por um bom período de tempo, o que pode ajudar a manter a memória da organização;

- Métodos de votação mais elaborados (que podem vir a ser mais consensuais) podem ser utilizados, já que a computação dos votos é feita de uma maneira automática;
- O resultado pode ser obtido imediatamente após o fim do término da votação, ou até mesmo parcialmente durante o período de votação;
- Mensagens de *e-mail* podem alertar aos usuários que ainda não votaram sobre um período de votação que está perto de se encerrar.

Acreditamos que a implentação de um sistema **vIBIS** em uma organização pode melhorar a qualidade das discussões e deliberações por causa dos vários benefícios apresentados neste trabalho. Acreditamos, entretanto, que do ponto de vista prático, tais benefícios venham a ser incorporados de uma maneira gradual. Isto é, inicialmente os usuários achariam o sistema conveniente, já que ele dispensa as pessoas de se reunirem sempre que tiverem de tomar uma decisão. Em um segundo momento, os usuários passariam a estender a noção de eleições e democracia, e a experimentar diferentes métodos de votação, sempre buscando aqueles que lhes parecessem mais consensuais.

Podemos visualizar como extensão ou melhoramento deste trabalho os seguintes tópicos:

- Análise sociológica da utilização de um sistema **vIBIS** dentro de situações reais. Dentro dessa análise, achamos que seria bastante interessante, não só levar em conta o fato de se usar uma ferramenta eletrônica para a interação, mas também analisar o impacto da possibilidade de uso de diferentes métodos de votação.
- Análise do tráfego de informações em uma rede **vIBIS**.
- Modelagem da implementação para uma base de dados distribuída. As decisões poderiam estar armazenadas em diferentes *sites* de maneira replicada (ou não), o que poderia agilizar o acesso às informações.
- Incorporação de objetos multimídia como anexos. Os objetos não-**IBIS** poderiam armazenar qualquer coisa, como, por exemplo, um desenho ou uma trilha de som e vídeo.
- Implementação de servidores de "métodos de seleção de vencedores", como sugerimos no Capítulo 4.

- Novas implementações de sistemas **vIBIS**, inclusive em diferentes plataformas.

Anexo I

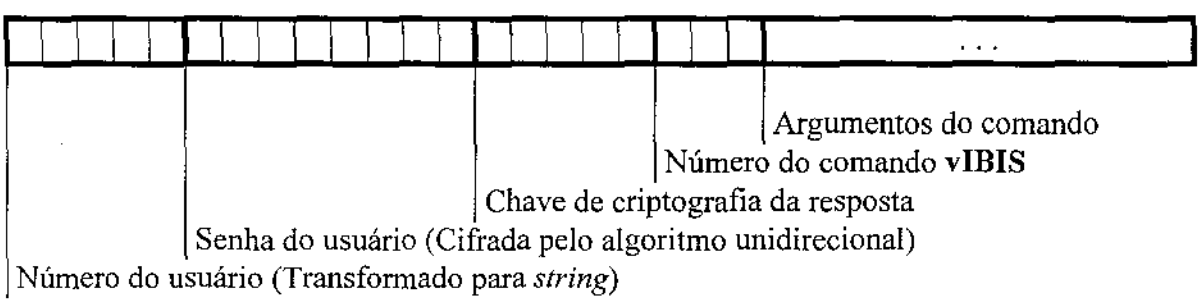
Protocolo vIBIS de comunicação

Como o sistema **vIBIS** de discussão e votação é um sistema desenvolvido baseado na metodologia cliente/servidor, os dois módulos (cliente e servidor) têm que trocar mensagens entre si. Surgiu então a necessidade de criar um formato para essas mensagens, e criar um protocolo de comunicação entre as duas partes. Neste anexo descrevemos o formato das mensagens que trafegam nos dois sentidos (sentido cliente-servidor e servidor-cliente) e também os vários comandos e mensagens de erros que devem ser suportados por clientes e servidores **vIBIS**.

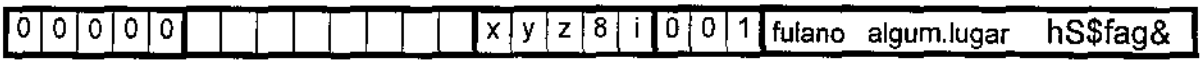
Todas as mensagens direcionadas a um servidor têm que ser cifradas segundo a chave pública desse servidor (ver capítulo sobre implementação). A resposta do servidor para esse cliente é cifrada pela chave de cinco bytes remetida pelo cliente dentro do corpo da mensagem. O cliente deve, então, gerar a cadeia de caracteres que formam a mensagem, cifrá-la, e só então remetê-la ao servidor.

Mensagens no sentido cliente-servidor

Mensagens no sentido cliente-servidor são sempre pedidos de *login*, consulta, alteração, etc., que o módulo cliente faz ao módulo servidor. Tais mensagens obedecem ao seguinte formato:

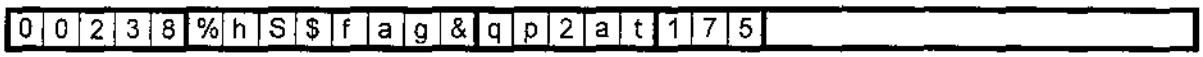


Por exemplo, a primeira mensagem que o módulo cliente deve remeter ao servidor é um pedido de *login* do usuário (comando **vIBIS** número 1). Tal mensagem para o usuário *fulano@algum.lugar* (número 238 no servidor em questão), cuja senha seja "passwd" (cifrada equivale a "hS$fag&") seria:



Nesse caso, o módulo cliente não está *logado* a nenhum usuário, portanto o número do usuário que está remetendo o comando é zero e sua senha vazia. A partir do momento que o servidor confirma o *login* deste usuário, todas as mensagens passam a conter seu número e senha nos devidos campos.

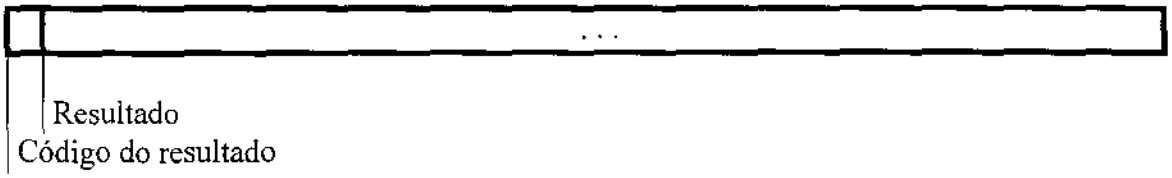
Uma mensagem deste usuário com o comando de leitura da lista das questões (comando **vIBIS** número 175) ficaria da seguinte maneira:



Nesse caso, o comando não contém argumentos.

Mensagens no sentido servidor-cliente

Sempre que um servidor recebe uma mensagem do cliente, ele tem que remeter uma resposta, que tanto pode ser o resultado do pedido quanto uma mensagem de erro. O formato da mensagem retorno é:



Código do resultado é um *byte* que pode conter um caracter de sucesso na operação (OK), ou um código de erro. Os códigos de resultado estão especificados mais adiante, neste mesmo Anexo.

Para o exemplo da mensagem pedindo o *login* para o usuário *fulano@algum.lugar*, citado na seção anterior, a mensagem retornada seria:

A	00238

O primeiro caracter "A" significa que a operação obteve sucesso, e o resultado é o código do usuário. No segundo exemplo, em que é solicitada uma lista das questões, a resposta seria a descrição das questões concatenadas (maiores detalhes na descrição do comando).

A	<issue 1> <issue 2> ... <issue *n*>

Código de resultados

Os tipos predefinidos de resultados que podem retornar de um servidor são apresentados na Tabela 28. O quê significa cada um deles depende do comando enviado ao servidor e será explicado na próxima seção. Os caracteres de "B" a "K" estão reservados para códigos de erros, que são remetidos do módulo cliente para o módulo interface. Por exemplo, se não for possível uma conexão com o servidor, o módulo cliente passa o erro correspondente. Os caracteres "a" a "z" estão disponíveis para novas mensagens de erros que alguma implementação queira criar. A ocorrência de um erro ou outro depende do tipo de comando enviado ao servidor, porém mensagens contendo o erro vERR_ACCESS_DENIED podem ocorrer para qualquer comando **vIBIS**. Por essa razão, omitimos esse tipo de "retorno" da tabela de descrição dos comandos (Tabela 29).

Resultado	Cod	Resultado	Cod
vOK	A	vERR_IN_GRP	S
vERR_WRONG_CMD	L	vERR_DONOT_EXIST	T
vERR_ACCESS_DENIED	M	vERR_CHILDREN	U
vERR_UNREGISTERED_US	N	vERR_FILLED	V
vERR_INVALID_PASSWORD	0	vERR_PERIOD	W
vERR_IS_MANAGER	P	vERR_WRONG_NUMBER	X
vERR_PROCESS_FAIL	Q	vERR_SAME_NAME	Y
vERR_NOT_IN_GRP	R		

Tabela 28 - Códigos de erros de mensagens servidor-cliente

Nome do Comando	Cod	Descrição	Parâmetros	No lado so servidor	Retorno
LOGIN	58	Checa login do usuário.	us.name+us.dom+ us.passwd (cifrada)	O servidor confere a senha do usuário. Se o usuário estiver cadastrado e a senha correta, o retorno é o número do usuário e a última vez que ele se logou.	vOK+us.id+us.day vERR_INV_LOGIN
NOTHING	59	Comando sem nenhuma ação.		Esse comando serve para o cliente checar se o servidor está ativo.	vOK
INC_US	60	Solicita a inclusão de um usuário. O usuário só fará acesso após receber a senha por *e-mail*.	us.name+us.dom	O servidor deve checar a existência do usuário antes de cadastrá-lo. No momento do cadastro, uma senha deve ser sorteada e remetida ao usuário via *e-mail*.	vOK+us.id vERR_US_EXISTS
READ_US_ID	61	Dado o nome do usuário, pede seu número identificador no servidor.	us.name+us.dom		vOK+us.id vERR_DONOT_EXISTS
READ_USNAME	62	Dado o número do usuário, retorna o nome e domínio.	us.id		vOK+us.name+dom.name vERR_DONOT_EXISTS
READ_US_GRP_LIST	63	Fornece a lista de todos os grupos que o usuário faz parte.			vOK+lista(grp.id+grp.name+ manager.name+manag.dom)
CHANGE_PASS	64	Solicita a mudança de senha do usuário.	oldpass(cifrada)+ newpass(não cifrada)	O servidor muda a senha do usuário e, dependendo dos parâmetros de mail do usuário, remete a nova senha por *e-mail*.	vOK
READ_DOM_NAME	70	Dado o número do domínio, retorna o nome.	dom.id		vOK+dom.name vERR_DONOT_EXISTS
READ_DOM_ID	71	Dado um nome de domínio, retorna o número.	dom.name		vOK+dom.id vERR_DONOT_EXISTS
READ_DOM_LIST	72	Fornece uma lista dos domínios cadastrados no servidor.			vOK+lista(dom.id+dom.name)
READ_DOM_US_LIST	73	Fornece uma lista dos usuários de um domínio (identificado pelo número).	dom.id		vOK+lista(us.name+us.dom)
READ_DOM_GRP_LIST	75	Fornece uma lista dos grupos acessíveis a partir de um determinado domínio identificado.	dom.id	Retorna número e nome do grupo, juntamente com o endereço do gerente e o tipo de entrada através do domínio no grupo (fechada/aberta).	vOK+lista(grp.id+grp.name+ manager+entry)
READ_GRP	116	Dado o número, retorna os dados do grupo.	grp.id		vOK+grp.id+grp.name+ manager
READ_GRP_LIST	117	Lista todos os grupos que o usuário participa ou poderá ter acesso através de seu domínio.			vOK+lista(grp.id+grp.name+ manager)
READ_ALLGRP_LIST	118	Lista todos os grupos cadastrados no servidor.			vOK+lista(grp.id+grp.name+ manager)
READ_OPEN_GRP_LIST	119	Fornece o número dos grupos que são abertos para o usuário.			vOK+lista(grp.id)

Tabela 29 - Descrição dos comandos vIBIS

Nome do Comando	Cod	Descrição	Parâmetros	No lado so servidor	Retorno
READ_GRP_DOM_LIST	120	Fornece os domínios que podem ter acesso ao grupo.	grp.id	O usuário deve fazer parte do grupo para ter essa informação.	vOK+lista(dom.id+dom.name+entry)
READ_GRP_US_LIST	121	Fornece a lista dos usuários que fazem parte do grupo.	grp.id	O usuário deve fazer parte do grupo.	vOK+lista(us.id+us.name+us.dom)
READ_GRP_DOM_US_LIST	122	Fornece a lista de todos os usuários dos domínios que podem ter acesso ao grupo.	grp.id	O usuário deve fazer parte do grupo.	vOK+lista(us.id+us.name+us.dom)
READ_GRP_ISS_LIST	123	Fornece a lista de todas as questões às quais o grupo tem acesso.	grp.id	O usuário deve fazer parte do grupo. O servidor deve retornar as informações sobre a questão e o direito do grupo (vote,voice,watch).	vOK+lista(iss.id+iss.name+manager+right)
CREATE_GRP	128	Solicita a criação de um grupo de discussão.	grp.name	O servidor cria o grupo com entrada fechada no domínio do usuário, inclui o usuário no grupo e coloca-o como gerente.	vOK+grp.id vERR_PROCESS_FAIL
INS_GRP_DOMs	130	Dá acesso a domínios em um grupo.	grp.id+entry+lista(dom.id)	O usuário deve ser o gerente do grupo.	vOK+lista(dom.id+dom.name+entry)
DEL_GRP_DOMs	131	Tira o acesso dos domínios no grupo.	grp.id+lista(dom.id)	O usuário deve ser o gerente do grupo.	vOK+lista(dom.id+dom.name)
SET_GRP_DOMs_ENTRY	132	Modifica o tipo de entrada para domínios em um grupo.	grp.id+entry+lista(dom.id)	O usuário deve ser o gerente do grupo.	vOK+lista(dom.id+dom.name+entry)
INS_GRP_DOMNAMEs	133	O mesmo que INS_GRP_DOMs, porém o identificador dos domínios é o próprio nome.	grp.id+entry+lista(dom.name)		vOK+lista(dom.id+dom.name+entry)
INS_GRP_USs	136	Insere usuários em um grupo.	grp.id+lista(us.id)	O usuário que envia o comando deve ser o gerente do grupo, ou o próprio usuário, caso a entrada no domínio do usuário seja aberta. Os usuários que não existirem não farão parte do retorno, logo o retorno pode ser simplesmente um vOK.	vOK+lista(us.id+us.name+us.dom)
DEL_GRP_USs	137	Elimina usuários de um grupo.	grp.id+lista(us.id)	O usuário que envia o comando deve ser o gerente, ou um usuário pedindo sua própria exclusão. Os usuários inexistentes não farão parte do retorno.	vOK+lista(us.id+us.name+us.dom)
INS_GRP_USNAMEs	138	O mesmo que INS_GRP_USs.	grp.id+lista(us.name+us.dom)	O usuário deve ser o gerente. Os usuários que não existirem serão cadastrados no servidor. A senha será sorteada e enviada por *e-mail* ao usuário.	vOK+lista(us.id+us.name+us.dom)
DEL_GRP_USNAMEs	139	O mesmo que DEL_GRP_USs.	grp.id+lista(us.name+us.dom)	O usuário deve ser o gerente.	vOK+lista(us.id+us.name+us.dom)
SET_GRP_MAN	140	Transfere os direitos de gerente de um grupo para outro usuário.	us.id	O usuário que envia o comando deve ser o gerente do grupo; o outro usuário deve já fazer parte do grupo.	vOK+us.id+us.name+us.dom vERR_NOT_IN_GRP
SET_GRP_MANNAME	141	O mesmo que SET_GRP_MAN.	us.name+us.dom		vOK+us.id+us.name+us.dom

Tabela 29 - Descrição dos comandos vIBIS (Continuação)

Nome do Comando	Cod	Descrição	Parâmetros	No lado so servidor	Retorno
DROP_GRP	142	Elimina grupo de discussão.	grp.id	O usuário deve ser o gerente; o grupo deve estar sem nenhum usuário, a não ser o gerente, e nenhuma questão pode ter sido aberta para esse grupo.	vOK+grp.id vERR_NOT_EMPTY
INS_GRP_USs_FROM_DOM	143	Solicita a inclusão no grupo de todos os usuários do domínio.	grp.id+dom.id	O usuário precisa ser o gerente. O servidor retorna vERR_NOT_EXISTS caso dom.id não exista.	vOK+dom.id vERR_NOT_EXISTS
INS_GRP_USs_FROM_GRP	144	Solicita a inclusão dos usuários pertencentes a outro grupo.	grp1.id+grp2.id	O usuário precisa ser gerente do grupo destino (grp1).	vOK+grp2.id
READ_ISS	174	Fornece informações sobre uma questão.	iss.id	O servidor deve checar se o usuário tem acesso à questão. Caso a questão seja uma sub-questão, o campo iss.parent traz o nó pai.	iss.parent+iss.id+iss.header+manager+nrnds+anony+secret+partial+change+nodes + cur_rnd+dat_disc+dat_vot1+dat_vot2
READ_ISS_LIST	175	Fornece uma lista das questões que o usuário tem acesso.		Como as questões estão dispostas em forma de árvore, o campo iss.parent traz o número do nó pai de cada questão. Os campos max_arg, max_ax trarão os últimos valores para posições, argumentos e anexos. Essa informação pode ser usada pela interface para informar ao usuário sobre novas contribuições.	iss.parent+iss.id+iss.header+manager+right+cur_rnd+dat_disc+dat_vot1+dat_vot2+max_pos+max_arg+max_ax
READ_ISS_TEXT	176	Fornece o texto de uma questão.	iss.id	O usuário deve ter acesso à questão.	iss.tex
READ_ISS_POS_LIST	177	Fornece a lista das posições de uma questão.	iss.id	O usuário deve ter acesso à questão.	vOK+lista(pos.parent,pos.id,pos.header,author,pos_rnd)
READ_ISS_GRP_LIST	178	Fornece a lista dos grupos que possuem acesso à questão.	iss.id	O usuário deve ter acesso à questão.	vOK+lista(grp.id+grp.name+manager+right)
READ_VOTED_ISS_LIST	180	Fornece os números das questões às quais o usuário já votou no turno corrente.			vOK+lista(iss.id)
READ_ISS_RND_LIST	181	Fornece as informações sobre todos os turnos de uma questão.	iss.id	O usuário deve ter acesso à questão. O servidor retorna os dados referentes a datas e formas de votação e seleção do vencedor.	vOK+lista(rnd+dat_disc+dat_vot1+dat_vot2+vot+wsm+unt,n)
WRITE_ISS_TEXT	182	Solicita a alteração do texto da questão.	iss.id+iss.tex	O usuário deve ser o gerente da questão.	vOK vERR_PROCESS_FAIL
CREATE_ISS	183	Solicita a criação de uma nova questão, a ser discutida pelo grupo "grp.id"	iss.parent+iss.header+grp.id+iss.tex	Caso o campo iss.parent contenha algum número, o servidor cria uma sub-questão com os valores das propriedades da questão iguais à questão pai. Nesse caso o campo grp.id é ignorado e os grupos dessa questão serão os mesmo da questão pai. O gerente da questão é o mesmo gerente do grupo remetido (caso não seja uma sub-questão).	vOK+iss.parent+iss.id+iss.header+manager+right

Tabela 29 - Descrição dos comandos vIBIS (Continuação)

Nome do Comando	Cod	Descrição	Parâmetros	No lado so servidor	Retorno
INS_ISS_GRPs	184	Solicita a inclusão de grupos de discussão em uma questão.	iss.id+right+lista(grp.id)	O usuário deve ser o gerente da questão.	vOK+lista(grp.id+grp.name+manager+right)
DEL_ISS_GRPs	185	Solicita a exclusão de grupos de uma questão.	iss.id+lista(grp.id)	O usuário deve ser o gerente da questão. Caso algum grupo não exista, ou já não faça parte da questão, seu número não fará parte do retorno.	vOK+lista(grp.id)
SET_ISS_GRPs	186	Altera direitos de grupos em uma questão.	iss.id+right+lista(grp.id)	O usuário deve ser gerente da questão.	vOK+lista(grp.id+grp.name+manager+right)
SET_ISS_MAN	189	Solicita alteração do gerente da questão.	iss.id+us.id	O usuário que envia o comando deve ser o gerente da questão.	vOK+us.id+us.name+us.dom
SET_ISS_MANNAME	190	O mesmo que SET_ISS_MAN.	iss.id+us.name+us.dom		vOK+us.id+us.name+us.dom
DROP_ISS	191	Elimina questão.	iss.id	O usuário deve ser o gerente da questão. A questão não pode conter nenhuma contribuição (posições, argumentos ou anexos).	vOK+iss.id vERR_NOT_EMPTY
SET_ISS_PROP	192	Solicita alteração das propriedades de uma questão.	iss.id+nmds+anony+secret+partial+change+nodes	O usuário deve ser gerente da questão, e esta não pode estar no meio de um processo de votação.	vOK+nmds+anony+secret+partial+change+nodes
SET_ISS_RND	193	Solicita a alteração de um turno de uma questão.	iss.id+rnd+dat_disc+dat_vot1+dat_vot2+vot+wsm+unt+n	O usuário deve ser o gerente e o número do turno não pode ultrapassar o número de turnos. Caso o turno não exista, deve ser inserido com os valores especificados.	vOK+rnd+dat_disc+dat_vot1+dat_vot2+vot+wsm+unt+n vERR_WRONG_NUMBER
READ_ISS_RND	194	Fornece os dados relativos a um turno de uma questão.	iss.id [+rnd]	O usuário deve ter acesso à questão. Caso o segundo parâmetro seja omitido, ou passado um número de turno que não exista, o retorno será relativo ao turno corrente da questão.	vOK+rnd+dat_disc+dat_vot1+dat_vot2+vot+wsm+unt+n
DROP_ISS_RND	195	Solicita a exclusão de um turno de uma questão.	iss.id+rnd	O usuário deve ser gerente, o turno deve ser posterior ao turno corrente e não pode haver nenhum turno posterior a este.	vOK+rnd vERR_WRONG_NUMBER
READ_POS	232	Fornece informações sobre uma posição.	iss.id+pos.id	O usuário deve ter acesso à questão.	vOK+pos.parent+pos.id+pos.header+author
READ_POS_TEXT	233	Fornece o texto de uma posição.	iss.id+pos.id	O usuário deve ter acesso à questão.	vOK+pos.tex
READ_POS_ARG_LIST	234	Fornece a lista dos *argumentos* relacionados com uma posição.	iss.id+pos.id	O usuário deve ter acesso à questão.	vOK+lista(arg.id+flag+arg.header+author)
WRITE_POS_TEXT	235	Altera um texto de uma posição.	iss.id+pos.id+pos.tex	O usuário deve ser o autor da posição, a posição deve estar ativa no turno corrente e o período de discussão não pode ter-se encerrado.	vOK vERR_PERIOD vERR_WRONG_POS
INS_POS	236	Solicita a inclusão de uma posição.	iss.id+pos.parent+pos.header+anony+pos.tex	O usuário deve ter direito de voz na questão, o período de discussão não pode ter-se esgotado e a posição pai deve estar ativa no turno. O campo anony será ignorado caso a questão não suporte contribuições anônimas.	vOK+pos.id vERR_PERIOD vERR_WRONG_POS

Tabela 29 - Descrição dos comandos vIBIS (Continuação)

Nome do Comando	Cod	Descrição	Parâmetros	No lado so servidor	Retorno
DEL_POS_ARG	237	Solicita a exclusão da relação entre um argumento e uma questão.	iss.id+pos.id+arg.id	O usuário deve ser autor do argumento, o período de discussão não pode ter-se esgotado e a posição deve estar ativa no turno corrente.	vOK+arg.id vERR_PERIOD vERR_WRONG_POS
DEL_POS	238	Solicita a exclusão de uma posição.	iss.id+pos.id	O usuário deve ser o autor da posição, o período de discussão não deve estar encerrado e a posição deve estar ativa no turno.	vOK+pos.id vERR_PERIOD vERR_WRONG_POS
READ_ARG_TEXT	290	Fornece o texto de um argumento.	iss.id+arg.id	O usuário deve ter acesso à questão.	vOK+arg.tex
READ_ARG_POS_LIST	291	Fornece a lista das posições relacionadas com um argumento.	iss.id+arg.id	O usuário deve ter acesso à questão.	vOK+lista(arg.id+flag+ arg.header+author.name)
READ_OFFARG_POS_LIST	292	Fornece a lista das posições que não estão relacionadas com um argumento.	iss.id+arg.id	O usuário deve ter acesso à questão.	vOK+lista(pos.id+pos.header+ author.name)
WRITE_ARG_TEXT	293	Altera o texto de um argumento.	iss.id+arg.id+arg.tex	O usuário deve ser o autor do argumento, o período de discussão não pode estar esgotado e o argumento deve estar relacionado a alguma posição ativa no turno.	vOK vERR_PERIOD vERR_WRONG_POS
INS_ARG	294	Solicita a inclusão de um argumento.	iss.id+pos.id+flag+ arg.header+anony+ arg.tex	O usuário deve ter direito a voz na questão, deve estar dentro do período de discussão e a posição relacionada deve estar ativa no turno.	vOK+arg.id vERR_PERIOD vERR_WRONG_POS
INS_ARG_POSs	295	Relaciona posições a um argumento.	iss.id+arg.id+flag+ lista(pos.id)	O usuário deve ser autor do argumento e o período de discussão não pode ter-se encerrado. As posições que não estiverem ativas no turno serão ignoradas e não farão parte do retorno.	vOK+lista(pos.id) vERR_PERIOD
DEL_ARG_POSs	296	Elimina o relacionamento de posições com um argumento.	iss.id+arg.id+lista(pos.id)	O usuário deve ser o autor do argumento e o período de discussão não pode ter-se encerrado. As posições não ativas serão ignoradas e não farão parte do retorno.	vOK+lista(pos.id) vERR_PERIOD
READ_ISS_AX_LIST	348	Lista dos anexos de uma questão.	iss.id	O usuário deve ter acesso à questão.	vOK+lista(ax.id+ax.header)
READ_AX_TEXT	349	Fornece o texto de um anexo.	iss.id+ax.id	O usuário deve ter acesso à questão.	vOK+ax.tex
INS_AX	350	Solicita a inclusão de um anexo.	iss.id+pos.id+arg.id+ ax.header+anony+ax.tex	O usuário deve ter direito a voz na questão e o período de discussão não pode ter-se encerrado. Caso a questão não suporte contribuições anônimas, o campo anony será ignorado. Se o anexo não for ligado a uma posição ou argumento, os campos correspondentes deverão estar em branco.	vOK+ax.id vERR_PERIOD
DEL_AX	351	Solicita a exclusão de um anexo.	iss.id+ax.id	O usuário deve ser o autor do anexo e o período de discussão não pode ter-se encerrado.	vOK+ax.id vERR_PERIOD
WRITE_AX_TEXT	352	Solicita a alteração do texto de um anexo.	iss.id+ax.id+ax.tex	O usuário deve ser o autor do anexo e o período de discussão não pode ter-se encerrado.	vOK vERR_PERIOD

Tabela 29 - Descrição dos comandos vIBIS (Continuação)

Nome do Comando	Cod	Descrição	Parâmetros	No lado so servidor	Retorno
READ_US_VOT	406	Solicita os votos do usuário em um turno de uma questão.	iss.id+rnd	Caso o usuário não tenha votado no turno, o retorno será vazio (vOK+"").	vOK+lista(pos.id+vot.value)
WRITE_US_VOT	407	Remete os votos do usuário no turno corrente de uma questão.	iss.id+lista(pos.id+ vot.value)	Deve estar dentro do período de votação e o usuário deve ter direito a voto na questão. As posições que não estiverem ativas deverão ser ignoradas nas rotinas de seleção dos vencedores. O servidor não faz consistência dos votos.	vOK+lista(pos.id+vot.value) vERR_PERIOD
DROP_VOTs	408	Solicita o cancelamento dos votos do usuário.	iss.id	Deve estar dentro do período de votação e o parâmetro de mudança de votos da questão deve estar ativo.	vOK vERR_PERIOD
READ_ISS_VOT	409	Fornece os votos de todos os usuários em uma questão.	iss.id+rnd	O usuário deve ter acesso à questão e ou o período de votação deve ter-se encerrado ou o parâmetro de resultados parciais estar ativo. Caso os votos sejam secretos, o us.id será do usuário "anonymous".	vOK+lista(us.id+lista(pos.id+ vot.value)+"zzz") vERR_PERIOD
READ_RND_RESULT	410	Fornece o resultado de um turno em uma questão.	iss.id+rnd	O usuário deve ter acesso à questão. Caso o resultado do turno ainda não tenha sido gravado pelo gerente, o retorno será vazio (vOK+"").	vOK+lista(pos.id+pos.ord)
WRITE_RND_RESULT	411	Solicita a gravação do resultado do turno corrente de uma questão.	iss.id+lista(pos.id+ pos.grid)	O usuário deve ser o gerente da questão e o período de votação deve ter-se encerrado. As posições que tiverem seu pos.grid menor ou igual ao número de ganhadores no turno (n), passarão ativas ao turno posterior.	vOK vERR_PERIOD
CLOSE_RND	416	Solicita o fechamento do turno corrente.	iss.id	O usuário deve ser o gerente da questão e os resultados do turno já devem estar gravados. O servidor ativará as posições que passarão para o próximo turno, segundo os resultados gravados.	vOK vERR_EMPTY
DROP_RESULT	417	Solicita a eliminação dos resultados gravados do turno corrente.	iss.id	O usuário deve ser o gerente.	vOK

Tabela 29 - Descrição dos comandos vIBIS (Continuação)

Anexo II

Base de dados vIBIS

Neste anexo, apresentamos as tabelas da base de dados de um servidor **vIBIS**. Essas tabelas devem ser utilizadas como referência para análise dos comandos do protocolo descritos no Anexo I. No protótipo do servidor desenvolvido utilizamos o SGBD **Sybase** [SYB92A] como ferramenta de armazenamento de dados. O servidor **vIBIS** implementado funciona como um programa cliente do servidor SQL, fazendo conexão através de rotinas de integração com o SGBD que estão disponíveis em uma biblioteca para linguagem C [SYB92B]. A coluna "Relacionamentos" indica a que tabela o campo faz referência, por exemplo, o campo **dom** da tabela **us** é uma referência ao campo **id** da tabela **dom**.

Tabela:	**dom**	**Domínios de Internet**

Campo	Tipo	Descrição	Relacionamentos
id	char(5)	identificador	
name	char(40)	nome do domínio	

Tabela:	**us**	**Usuários**

Campo	Tipo	Descrição	Relacionamentos
id	char(5)	identificador	
name	char(8)	*username* do usuário em seu domínio	
dom	char(5)	id do domínio	dom.id
passwd	char(8)	senha criptografada	
day	smalldatetime	data e hora do último *login*	
mail	byte	opções de mensagens automáticas	

Tabela:	grp	Grupos de Discussão

Campo	Tipo	Descrição	Relacionamentos
id	char(5)	identificador	
name	char(25)	nome do grupo	
manager	char(5)	id do usuário gerente	us.id

Tabela:	grp_dom	Domínios que tem acesso a Grupos

Campo	Tipo	Descrição	Relacionamentos
grp	char(5)	id do grupo	grp.id
dom	char(5)	id do domínio	dom.id
entry	boolean	entrada: (0 - fechado; 1 - aberto)	

Tabela:	grp_us	Usuários dos Grupos

Campo	Tipo	Descrição	Relacionamentos
grp	char(5)	id do grupo	grp.id
us	char(5)	id do usuário	us.id

Tabela:	iss	Questões (*Issues*)

Campo	Tipo	Descrição	Relacionamentos
id	char(5)	identificador	
parent	char(5)	questão que deu origem (se sub-*issue*)	iss.id
header	char(30)	nome da questão	
manager	char(5)	id do usuário gerente da questão	us.id
nrnds	int	número de turnos	
anony	boolean	contribuições anônimas	
secret	boolean	votos secretos	
partial	boolean	resultados parciais	
change	boolean	mudança de voto	
cur_rnd	int	turno corrente	
nodes	int	nós de votos (0)todos (1)nível (2)folhas	
tex	text	texto da questão	

Tabela:	iss_grp	Grupos que tem acesso a Questões

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
grp	char(5)	id do grupo	grp.id
right	int	direitos (0)assistir (1)voz (2)voto	

Tabela:	iss_rnd	Turnos das Questões

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
rnd	int	número do turno	
dat_disc	date	data limite para contribuições	
dat_vot1	date	início do período de votação	
dat_vot2	date	fim do período de votação	
vot	char(10)	forma de votação	
wsm	char(10)	método de seleção do vencedor	
unt	char(10)	método de desempate	
winners	int	número de vencedores	
man_res	text	considerações do gerente p/ o resultado	

Tabela:	pos	Posições (*Positions*)

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
id	char(3)	identificador	
parent	char(3)	posição que deu origem (se sub-pos)	pos.id
header	char(25)	nome da posição	
us	char(5)	id do usuário autor da posição	us.id
rnd	int	turno da posição	
tex	text	texto da posição	

Tabela:	arg	Argumentos (*Arguments*)

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
id	char(3)	identificador	
header	char(25)	nome do argumento	
us	char(5)	id do usuário autor do argumento	us.id
tex	text	texto do argumento	

Tabela:	pos_arg	Ligação Posições/Argumentos

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
pos	char(3)	id da posição	pos.id
arg	char(3)	id do argumento	arg.id
flag	char(1)	ligação (+)suporte (-)objeção (?)indif.	

Tabela:	ax	Anexos

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
id	char(3)	identificador	
header	char(30)	nome do anexo	
us	char(5)	id do usuário autor do argumento	us.id
tex	text	texto do anexo	

Tabela:	iss_ax	Ligações IPA/Anexos

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
pos	char(3)	id da posição	pos.id
arg	char(3)	id do argumento	arg.id
ax	char(3)	id do anexo	ax.id

Tabela:	vot	Votos

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
rnd	int	número do turno	iss_rnd.rnd
us	char(5)	id do usuário	us.id
pos	char(3)	id da posição	pos.id
value	int	valor do voto	

Tabela:	rnd_pos	Resultado nos turnos

Campo	Tipo	Descrição	Relacionamentos
iss	char(5)	id da questão	iss.id
rnd	int	número do turno	iss_rnd.rnd
pos	char(3)	id da posição	pos.id
ord	int	classificação da posição	

Tabela:	mail	Mensagens a serem remetidas

Campo	Tipo	Descrição	Relacionamentos
time	smalldatetime	hora do evento	
cmd	int	comando **vIBIS**	
par1	char(5)	1o parâmetro - *iss/pos/arg/grp...*	Dependem do comando
par2	char(5)	2o parâmetro	
us	char(5)	usuário ao qual deve ser remetido *mail*	us.id
line	char(100)	informações a constar na mensagem	
stat	char(1)	*status* - remetido/a remeter...	

Anexo III

Métodos de votação e seleção de vencedores

Segundo a tabela **iss_rnd** descrita no Anexo II, foram reservados três campos do tipo **char(10)** para armazenar forma de votação, métodos de seleção do vencedor e método de desempate. Nesses 10 caracteres, reservados para cada um desses itens, devem constar todas as informações sobre o método utilizado, incluindo sua identificação e parâmetros. Nos métodos já definidos, utilizamos o primeiro caracter para identificá-los e os demais para armazenar seus parâmetros.

A seguir apresentamos as diferentes formas de votação e, logo após, os métodos de seleção de vencedores.

Formas de votação

Standard

Campos	Tam	Descrição
id	1	Identificador = "A"
votos	2	número de votos permitidos
vetos	2	número de vetos permitidos

Exemplo:	Standard-2×1	A	0	2	0	1					

Approval

Campos	Tam	Descrição
id	1	Identificador = "B"
votos	2	número de votos permitidos (Se 0, indica número indefinido)

Exemplo:	Approval-10	B	1	0							

Ranking

Campos	Tam	Descrição
id	1	Identificador = "C"
posições	3	número máximo de posições (0 indica número indefinido)

Exemplo:	Ranking-0	C	0	0							

Métodos de seleção de vencedores p/ votação do tipo *Standard*

>(Vot -Vet)

Campos	Tam	Descrição
id	1	Identificador = "A"
peso voto	2	Peso para cada Voto
peso veto	2	Peso para cada Veto

Exemplo:	*>(Vot -Vet) 3×2*	A	0	3	0	2					

>Vot

Campos	Tam	Descrição
id	1	Identificador = "B"

Exemplo:	*>Vot*	B									

<Vet

Campos	Tam	Descrição
id	1	Identificador = "C"

Exemplo:	*<Vet*	C									

>(Vot -Vet)+N_vet out

Campos	Tam	Descrição
id	1	Identificador = "D"
peso voto	2	Peso para cada Voto
peso veto	2	Peso para cada Veto
N_vet_out	2	Número de quantas mais vetadas estarão fora

Exemplo:	*>(Vot -Vet)+N_vet out 1×1×2*	D	0	1	0	1	0	2			

>(Vot -Vet)+N_vot in

Campos	Tam	Descrição
id	1	Identificador = "E"
peso voto	2	Peso para cada Voto
peso veto	2	Peso para cada Veto
N_vot_in	2	Número de quantas mais votadas estarão dentro

Exemplo:	*>(Vot -Vet)+N_vot out 1×1×2*	E	0	1	0	1	0	2			

>(Vot -Vet)+N vot/vet in/out

Campos	Tam	Descrição
id	1	Identificador = "F"
peso voto	2	Peso para cada Voto
peso veto	2	Peso para cada Veto
N_vot_in	2	Número de quantas mais votadas estarão dentro
N_vet_out	2	Número de quantas mais vetadas estarão fora

Exemplo:	*>(... in/out 1×1×2×1)*	F	0	1	0	1	0	2	0	1	

>(Vot)+N_vet out

Campos	Tam	Descrição
id	1	Identificador = "G"
N_vet_out	2	Número de quantas mais vetadas estarão fora

Exemplo:	*>(Vot)+N_vet out 2*	G	0	2							

<(Vet)+N_vot in

Campos	Tam	Descrição
id	1	Identificador = "G"
N_vot_in	2	Número de quantas mais votadas estarão dentro

Exemplo:	*<(Vet)+N_vot in 3*	G	0	3							

Demais métodos

Outros métodos não parametrizados só são identificados pelo primeiro caracter. O surgimento de qualquer outro método deve utilizar sempre o espaço máximo de 10 caracteres, que foram reservados para a identificação completa do método, incluindo seus parâmetros. Os programas direcionadores de votação e de cálculo dos vencedores devem estar preparados para manipular as informações segundo as definições dos métodos. Caso uma interface não esteja preparada para a execução de um determinado método, deve haver a possibilidade de esta incluí-lo mediante o fornecimento dos programas e parâmetros do método.

Os demais métodos definidos no protótipo foram:

Método	**Identificador**
Approval	P
Borda	Q
Copeland	R
Ranking Multiplicativo	S
Fórmula I	T
Soma de Pesos	U
Média de Notas	V

Anexo IV

vIBIS Tool

Neste anexo, descrevemos a interface implementada durante o nosso curso de mestrado, à qual foi dada o nome de **vIBIS Tool**.

Para sua execução, o programa necessita de ambiente gráfico de alta resolução, colorido ou não; sistema operacional compatível com o sistema **UNIX**; e suportar a execução de um sistema gerenciador de janelas (*Window Manager*), que obedeça às definições utilizadas pelo ***Sistema X*** (*X Window System*) e que siga a especificação funcional ***OPENLOOK*** (*OPENLOOK Graphical User Interface Funcional Specification*) [SUN90B].

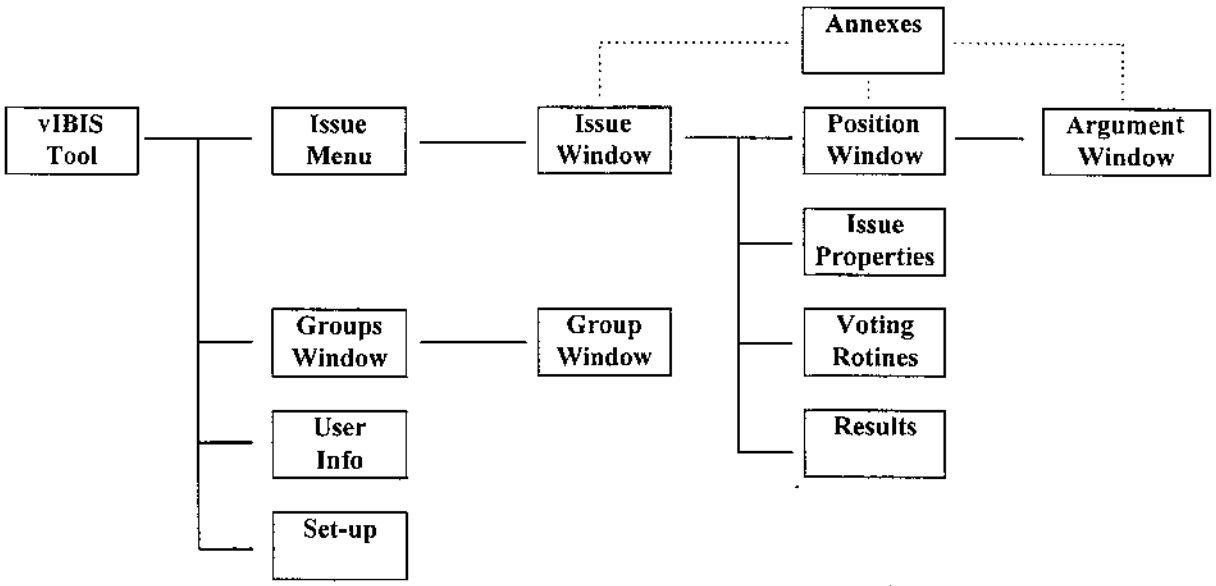


Figura 15 - Hierarquia de telas do vIBIS Tool

O sistema foi desenvolvido em estações de trabalho gráficas da **SUN**, que contém o gerenciador de janelas ***Open Windows***. Para o desenho dos objetos das

janelas do sistema foi utilizado o *toolkit* **Guide**, que gera código C/C++ para a biblioteca **Xview** [HEL93].

O sistema segue o padrão das interfaces WWW (Netscape, Mosaic), i.e., é aberta somente uma janela e nela são mostradas as diferentes telas do sistema. A parte superior da janela é armazenada para objetos genéricos a todas as telas, tais como: informações sobre o *login*; botões de *refresh* e saída do sistema; botão "menu" selecionador do *display* (esse seleciona dentro do nível 1 da hierarquia do sistema), e botões de avanço e retrocesso (<< e >>), nos quais o usuário pode clicar para avançar ou retroceder na hierarquia das rotinas já acessadas na execução. A hierarquia do sistema é organizada da forma como mostra a Figura 15. A parte variante da janela (parte central) é trocada, dependendo da rotina em execução. Algumas dessas rotinas são apresentadas a seguir:

Issue Menu

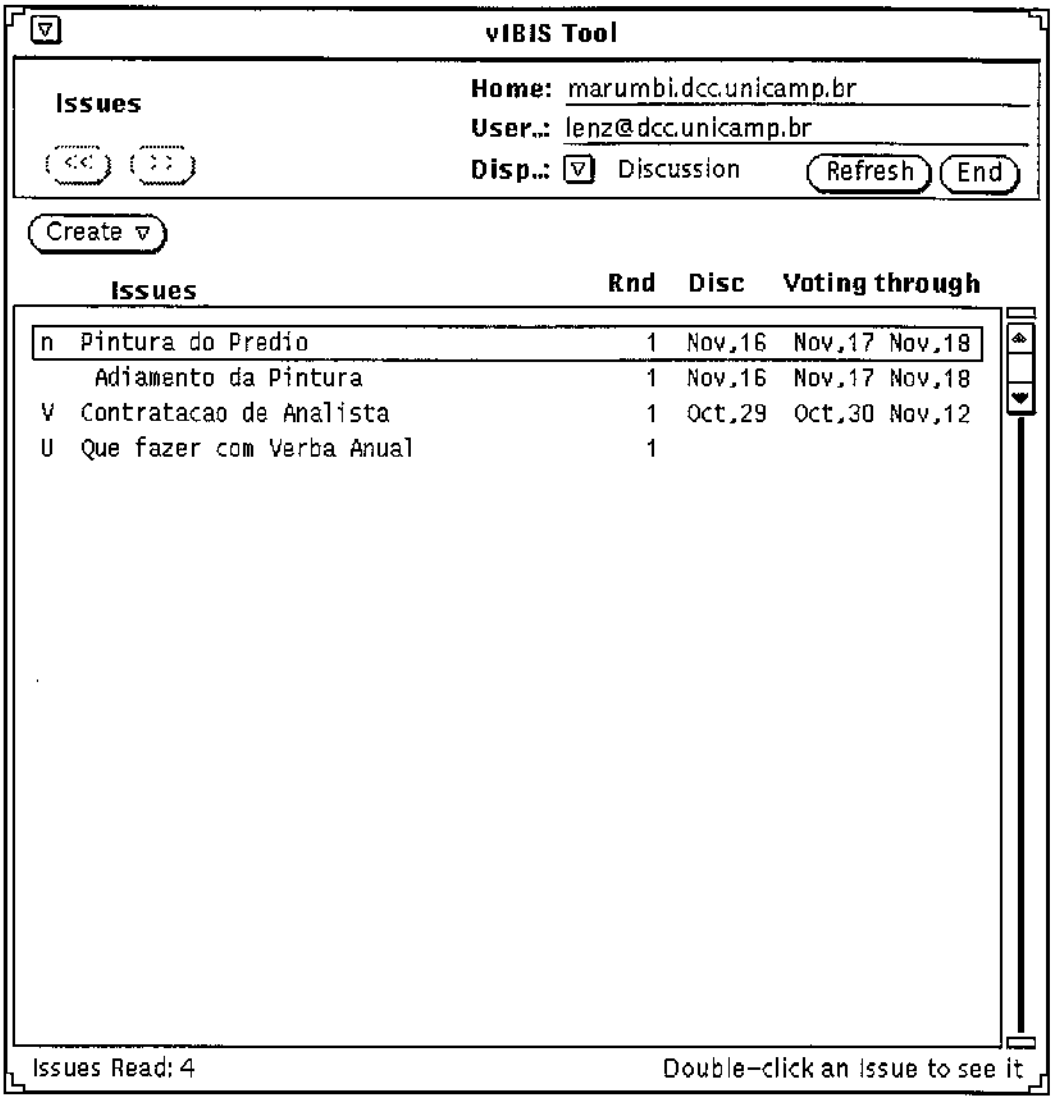


Figura 16 - Issue Menu

A Figura 16 tráz a primeira tela à qual o usuário tem acesso logo depois de efetuado o *login*. As informações apresentadas são uma lista das questões às quais o usuário *logado* tem acesso. As letras na primeira coluna dão algumas informações a respeito de cada questão: "N" indica que essa é uma questão nova; "n" que, na questão, existem contribuições novas; "U" indica que a questão ainda não foi lida pelo usuário; "V" que a questão está no período de votação e que o usuário ainda não votou; e "v" indica que o usuário já votou para a questão. Para ter acesso à questão desejada, o usuário dá um *double-click* na linha correspondente, e a tela da questão é aberta.

Issue Window

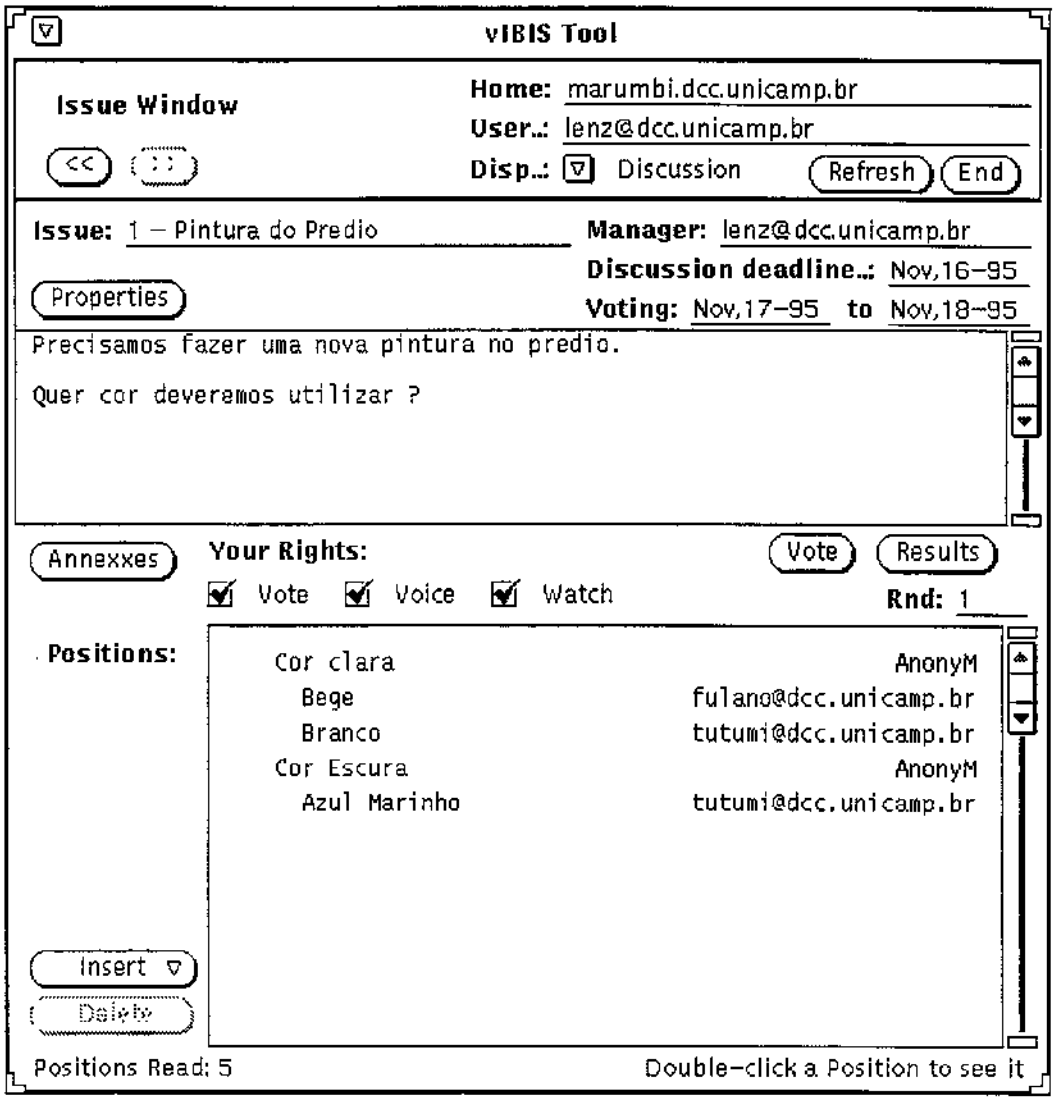


Figura 17 - Issue Window

Na *Issue Window* (Figura 17), o usuário tem acesso ao texto da questão e à lista de posições que respondem à questão. Para visualizar as propriedades da questão, tais como grupos de discussão que tem acesso à questão, métodos de votação nos turnos, etc; basta clicar no botão correspondente (*Properties*). Para expandir uma posição, dois cliques na linha correspondente fazem surgir a tela da posição. A partir dessa tela, também se tem acesso às rotinas de voto e resultados das votações.

Position Window

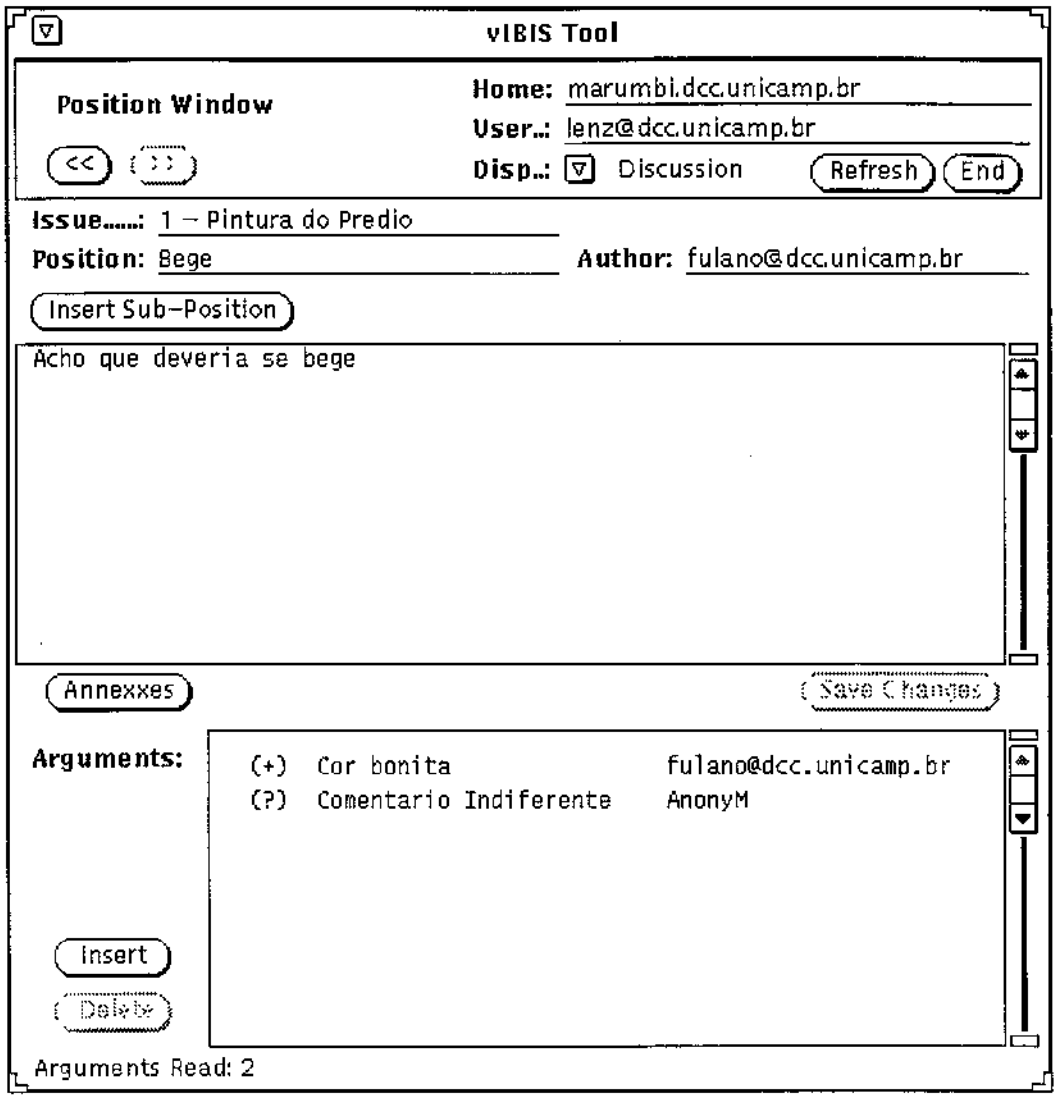


Figura 18 - Position Window

Nessa tela (Figura 18) é mostrado o texto da *position* selecionada e os argumentos que se ligam a ela. Argumentos podem ser de suporte, objeção ou indiferentes e são identificados pelos caracteres (+), (-) e (?). A partir dessa tela, pode-se abrir a tela de argumentos, que não descreveremos neste Anexo.

Voting Window

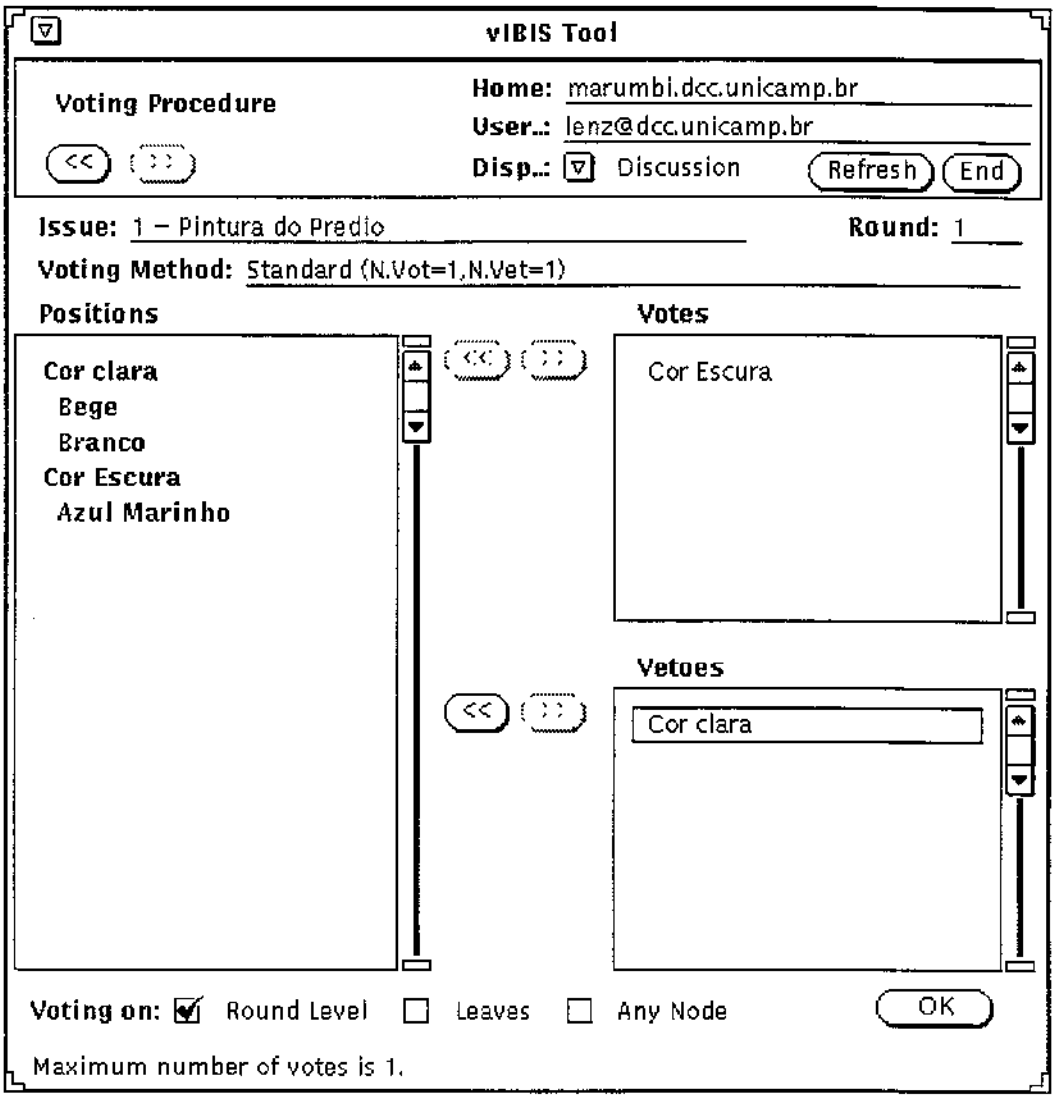


Figura 19 - Voting Window

Nessa interface foram desenvolvidas várias rotinas de voto que não precisam de programas externos para serem executadas. Tais rotinas fazem abrir uma tela para direcionar o voto dos usuários. No exemplo da Figura 19, vê-se uma votação do tipo *Standard-1×1*, na qual o usuário deve escolher uma *position* para voto e outra para veto. Outras formas de votação gerariam telas diferentes. Uma rotina de votação externa abriria uma outra janela *X* para capturar o voto do usuário.

vIBIS Managment System

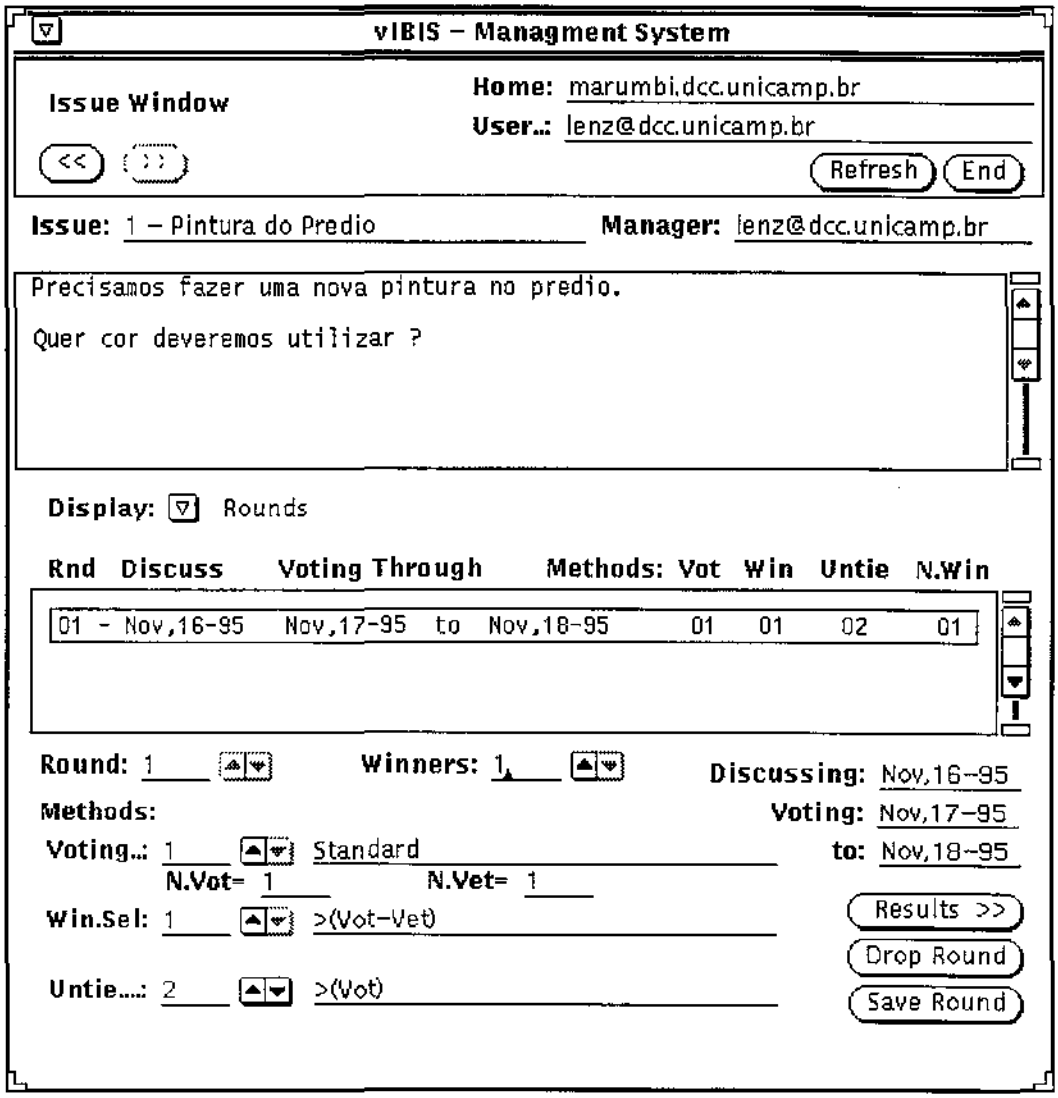


Figura 20 - vIBIS Managment System

Desenvolvemos as rotinas de gerenciamento de grupos e questões como um programa separado. Nesse programa, somente os gerentes têm acesso às questões e grupos. A partir dele, podem efetuar todas as operações de gerenciamento, tais como: incluir usuários em grupos; incluir domínios de ***INTERNET*** em grupos, incluir grupos em questões, definir propriedades de questões, definir períodos de discussão e votação de turnos, definir formas de votação e seleção de vencedores para os turnos de uma questão, etc. Na Figura 20 é apresentada a tela de configuração de turnos de uma questão.

Bibliografia

[OPP88] Susanna Opper. "A Groupware Toolbox" *Byte*, 275-282 (December 1988).

[ROB91] M.Robinson. "Computer supported co-operative work: cases and concepts", *Proceedings of Groupware '91* (1991).

[EGR91] C.A.Ellis, S.J.Gibbs and G.L.Rein. "Groupware - some issues and experiences", *Communications of the ACM*, (34) 1 (January 1991).

[JOH91] R. Johansen. "Leading Business Teams", Addison-Wesley, Reading, Mass (1991).

[KK88] Kenneth L. Kraemer and John Leslie King. "Computer-based systems for cooperative work and group decision making", *ACM Computing Surveys*, 20(2) (June 1988).

[CM88] Kevin Crowston and Thomas W. Malone, "Intelligent software agents", *Byte*, 267-272 (December 1988).

[EL88] Douglas Engelbart and Harvey Lehtman. "Working Together", *Byte*, 245-252 (December 1988).

[BS91] Liam J. Bannon and Kjeld Schmidt. "CSCW: Four characters in Search of a Context", *Studies in Computer Supported Cooperative Work*, Edited by J.M.Bowers and S.D.Benford, Elsevier Science Publishers B.V., North-Holland (1991).

[KL86] Brian Kantor and Phil Lapsley. "Network News Transfer Protocol - a proposed standard for the stream-based transmission of news". *Request for Comments: 977* (February 1986).

[KR70] W.Kunz and H.Rittel. "Issues as elements of information systems". Technical report, Institute of Urban and Regional Development, Univ. of California, Berkeley, CA (1970).

[CB88] Jeff Conklin and Michael L. Begeman. "gIBIS: A hypertext tool for exploratory policy discussion". *ACM Transaction on Office Information Systems*, 6(4): 303–331 (October 1998).

[YC90] K.C.Burgess Yakemovic and E.Jeffrey Conklin. "Report on a development project use of an issue-based information system". *CSCW 90: Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 105-118, Los Angeles (October 1990).

[CON92] E. Jeffrey Conklin, "Capturing Organizational Memory", *Groupware '92*, 133-137 (1992).

[RE91] G.L.Rein and C.A.Ellis. "rIBIS: A real-time group hypertext system". *Int. J. Man-Machine Studies*, 34(3) (1991).

[KSW93] Won Kim, Yongmoos Suh and Andrew B.Whinston, "An IBIS and object-oriented approach to scientific research data management", *J. Systems Software*, Elsevier Science Publishing Co., Inc., New York, 23:183-197 (1993).

[LL93] Shih-Hao Lee and Baw-Jhiune Liu, "IBO: an issue based object model for software design", *Proceedings of the 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering - Beijing*, 1: 270-274 (1993).

[IST89] "Emoções à vista", *Istoé Senhor*, Editora Três, São Paulo, **1048** (18/10/1989).

[MER88] Samuel Merril III, *"Making multicandidate elections more democratic"*, Princeton University Press, Princiton, New Jersey (1988).

[CON1788] Marquis de Condorcet, *"Essai sur la constitution et les fonctions des assemblées provinciales"*, Première partie (1788).

[BF82] Stevens J. Brams and Peter C. Fishburn. *"Approval Voting"*, Birkhäuser , Boston; Basel; Stuttgart (1992).

[BOR1781] J.C.Borda, "Memoire sur les Elections au Scrutin", *Histoire de l'Academie Royale de Science*, Paris (1781).

[CON74] Marquis de Condorcet, *"Mathématique et société"*, Choix de textes et commentaire par Roshdi Rashed, Hermann, Paris (1974).

[SCH86] Thomas Schwartz, *"The logic of collective choice"*, Columbia University Press, New York (1986).

[DP70] Frank DeMeyer and Charles R. Plott, "The probability of a cyclical majority", *Econometrica*, **38** (2) (1970).

[MOU83] H. Moulin, "The strategy of social choice", *Advanced Textbooks in Economics*, **18** (1983).

[KRA77] Gerald H. Kramer, "A dynamical model of political equilibrium", *Journal of Economic Theory*, **16**, 310-334 (1977).

[HAR1859] T.Hare, *"Treatise on the Election of Representatives, Parliamentary and Municipal"*, London, Longmans Green (1859).

[BUI87] Tung X. Bui. "Co-op: A group decision support system for cooperative multiple criteria group decision making", *Lecture Notes in Computer Science*, **290** (1987).

[SMP94] William E. Stein, Philip J. Mizzi and Roger C. Pfaffenberger, "A stocastic dominance analysis of ranked voting systems with scoring", *The European Journal of Operational Research*, 74(1) 78-85 (April 1994).

[LUC] William F. Lucas, "Social choice and decision making", *Introduction to Conteporary Mathematics*, 2nd edition, Part III, editor: Lynn A. Steen, W.H.Freeman and Company, New York.

[ARR63] K.J.Arrow, *"Social Choice and Indvidual Values"*, Yale University Press, New Haven (1963).

[BLAU72] Julian H. Blau, "A direct proof of Arrow's theorem", *Econometrica*, **40** (1) (1972).

[FISH74] Peter C. Fishburn, "Impossibility theorems without the social completeness axiom", *Econometrica*, **42** (4) (1974).

[KN74] Kiyoshi Kuga and Hiroaki Nagatani, "Voter antagonism and the paradox of voting", *Econometrica*, **42** (6) (1974).

[ROS75] Robert W. Rosenthal, "Voting Majority Sizes", *Econometrica*, **43** (2) (1975).

[DDH72] Otto A. Davis, Morris H. DeGroot and Melvin J. Hinich, "Social preference orderings and majority rule", *Econometrica*, **40** (1) (1972).

[SEN77] Amartya Sen, "Social choice theory: a re-rexamination", *Econometrica*, **45** (1) (1977).

[DEL91] Howard DeLong, *"A refutation of Arrow's theorem"*, University Press of America, Lanham, New York, London (1991).

[YOU74] H. P. Young, "An Axiomatization of Borda's Rule", *Journal of Economic Theory*, **9**, 43-52 (1974).

[CON92] E. Jeffrey Conklin, "Capturing Organizational Memory", *Groupware '92*, 133-137 (1992).

[CB88] Jeff Conklin and Michael L. Begeman. "gIBIS: A hypertext tool for exploratory policy discussion". *ACM Transaction on Office Information Systems*, 6(4): 303–331 (October 1998).

[BF82] Stevens J. Brams and Peter C. Fishburn. *"Approval Voting"*, Birkhäuser , Boston; Basel; Stuttgart (1992).

[MER88] Samuel Merril III, *"Making multicandidate elections more democratic"*, Princeton University Press, Princiton, New Jersey (1988).

[KL86] Brian Kantor and Phil Lapsley. "Network News Transfer Protocol - a proposed standard for the stream-based transmission of news". *Request for Comments: 977* (February 1986).

[SUN90A] Sun Microsystems, *"Network programming guide"* (1990).

[LUC86] Cláudio Leonardo Lucchesi, *"Introdução à criptografia computacional"*, Editora da UNICAMP, Campinas (1986).

[RSA78] R.L.Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, **21**, 120-6 (1978).

[KON81] Alan G. Konheim, *"Cryptography: a primer"*, John Wiley & Sons, Inc. (1981).

[HEL93] Dan Heller, *"XView Programming Manual for Xview Version 3.2"*, volume 7 of *The X window System Series*. O'Reilly & Associates, (Aug 1993).

[CPP92] Matthew Conway, Randy Pausch and Kimberly Passarella, *"A Tutorial for SUIT - The Simple User Interface Toolkit"*, Computer Science Department, University of Virginia (1992).

[CW94A] Flávio Lenz Cesar and Jacques Wainer, "vIBIS Tool - A groupware system for discussion and voting", *Anais do XXI Seminário Integrado de Software e Hardware*, Caxambu (1994).

[CW94B] Flávio Lenz Cesar and Jacques Wainer, "vIBIS: A discussion and voting sysem", *Journal of the Brazilian Computer Society*, **2** (1) (1994).

[SYB92A] SYBASE, *"Commands Reference Manual for SYBASE SQL server™ for UNIX"* (1992).

[SYB92B] SYBASE, *"Open Client DB-Library/C Reference Manual"* (1992).

[SUN90B] Sun Microsystems, *"OPENLOOK - Graphical User Interface Applications Style Guidelines"*. Addison-Wesley (June 1990).

[HEL93] Dan Heller, *"XView Programming Manual for Xview Version 3.2"*, volume 7 of *The X window System Series*. O'Reilly & Associates, (Aug 1993).