UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
PROJETO DE GRADUAÇÃO

RODRIGO LAIOLA GUIMARÃES

DESENVOLVIMENTO DA INFRAESTRUTURA COMPUTACIONAL DE GERENCIAMENTO E COMUNICAÇÃO DE DADOS DO AMBIENTE MULTIJADE

VITÓRIA
2004

RODRIGO LAIOLA GUIMARÃES

DESENVOLVIMENTO DA INFRAESTRUTURA COMPUTACIONAL DE GERENCIAMENTO E COMUNICAÇÃO DE DADOS DO AMBIENTE MULTIJADE

Dissertação apresentada como Projeto de Graduação do curso de Engenharia de Computação da Universidade Federal do Espírito Santo, como requisito para a obtenção do Grau de Engenheiro de Computação.
Orientador: Prof. Dr. Flávio Miguel Varejão.

VITÓRIA
2004

RODRIGO LAIOLA GUIMARÃES

DESENVOLVIMENTO DA INFRAESTRUTURA COMPUTACIONAL DE GERENCIAMENTO E COMUNICAÇÃO DE DADOS DO AMBIENTE MULTIJADE

COMISSÃO EXAMINADORA

Prof. Dr. Álvaro César Pereira Barbosa

Prof. Dr. Flávio Miguel Varejão
Orientador

Prof. Dr. José Gonçalves Pereira Filho

Vitória, 28 de outubro de 2004.

Dedico este trabalho a todos àqueles que acreditam que a ousadia e o erro são caminhos para as grandes realizações

AGRADECIMENTO

Agradeço a minha família pelo incentivo e pela compreensão durante toda minha vida e trajetória acadêmica.

Expresso meu agradecimento aos meus estimados professores da Universidade Federal do Espírito Santo pela ajuda nesta caminhada. Presto minha sincera admiração e respeito a todos os que sempre tiveram interesse e paciência em me ajudar.

Minha sincera gratidão e admiração pelo meu orientador Flávio Miguel Varejão por sua tamanha dedicação e incessante esforço em me ajudar durante o desenvolvimento deste projeto de pesquisa científica.

A Cláudia Schirley Braun e a Fernanda Assis Gama, sem as quais este trabalho não teria sido realizado.

Agradeço a todos os funcionários e alunos da UFES pela boa convivência que tivemos durante estes anos.

Acima de tudo e todos, agradeço a Deus por sempre ter iluminado meu caminho.

LISTA DE TABELAS

TABELA 1 - PERGUNTAS DO MÓDULO DE EXPLICAÇÃO DE SISTEMAS ADD----- 14
TABELA 2 - ENTIDADES DA ARQUITETURA MULTIADD ---------- 19
TABELA 3 - PRIMITIVAS DE SERVIÇO ---------- 37

LISTA DE FIGURAS

FIGURA 1 - *FRAMEWORK* OO USADO POR JADE ---------- 15
FIGURA 2 - INTERFACE GRÁFICA DE AQUISIÇÃO DO JADE ---------- 16
FIGURA 3 - INTERFACE DE EXPLICAÇÃO DO JADE. ---------- 16
FIGURA 4 - ARQUITETURA MULTIAGENTE PARA SISTEMAS ADD ---------- 19
FIGURA 5 - MESA VIRTUAL DE NEGOCIAÇÃO NO MULTIJADE ---------- 22
FIGURA 6 - PARÂMETROS DE COMUNICAÇÃO NO MULTIJADE ---------- 23
FIGURA 7 - FLUXOGRAMA DE MITIGAÇÃO DE UM CONFLITO ---------- 24
FIGURA 8 - MODELO DE INTERAÇÃO CENTRALIZADO ---------- 26
FIGURA 9 - CONSTRUÇÃO DE UM MODELO NO MULTIJADE ---------- 27
FIGURA 10 - MODELO E DOCUMENTO ATIVO NO MULTIJADE ---------- 28
FIGURA 11 - ARQUITETURA COMPUTACIONAL DO MULTIJADE ---------- 29
FIGURA 12 - INTERFACE DO SERVIDOR DE USUÁRIOS ---------- 30
FIGURA 13 - INTERFACE DO SERVIDOR DE MODELOS ---------- 32
FIGURA 14 - INTERFACE DO SERVIDOR CONTROLADOR ---------- 35
FIGURA 15 - TELA DE LOGIN DO MULTIJADE. ---------- 40
FIGURA 16 - TELA DE CRIAÇÃO DE UM MODELO NO MULTIJADE. ---------- 41
FIGURA 17 - DEFINIÇÃO DOS PARÂMETROS COMUNS AOS AGENTES ---------- 42
FIGURA 18 - DEFINIÇÃO DOS PARÂMETROS DE COMUNICAÇÃO ---------- 43
FIGURA 19 - DEFINIÇÃO DA RELAÇÃO DO PARÂMETRO DE COMUNICAÇÃO ---- 44
FIGURA 20 - REDE PARAMÉTRICA DO AGENTE RESPONSÁVEL PELA SUSPENSÃO. ---------- 45
FIGURA 21 - REDE PARAMÉTRICA DO AGENTE RESPONSÁVEL PELOS PNEUS 46
FIGURA 22 - INTERFACE DE CRIAÇÃO DE UM DOCUMENTO ATIVO ---------- 47
FIGURA 23 - INTERFACE DE ALTERAÇÃO DO VALOR DE PARÂMETRO ---------- 48
FIGURA 24 - SINALIZADOR DE NOVAS MENSAGENS ---------- 49
FIGURA 25 - MENSAGEM DE SUGESTÃO DE VALOR PARA PARÂMETRO --------- 50
FIGURA 26 - MENSAGEM DE NOTIFICAÇÃO DE CONFLITO ---------- 51
FIGURA 27 - MENSAGEM DE NOTIFICAÇÃO DE PERSISTÊNCIA DE CONFLITO-- 52
FIGURA 28 - ACESSO A TELA DE MODERAÇÃO ---------- 52
FIGURA 29 - INTERFACE DE MODERAÇÃO ---------- 53
FIGURA 30 - ARQUITETURA HIERÁRQUICA DO MULTIJADE ---------- 56
FIGURA 31 - ARQUITETURA MULTIJADE COM INTERAÇÃO TOTAL ---------- 57

SUMÁRIO

RESUMO ---- 8
1 - INTRODUÇÃO ---- 9
2 - DOCUMENTOS ATIVOS DE DESIGN ---- 12
2.1 - JADE ---- 14
2.2 - Design concorrente e rationale ---- 17
2.2.1 - MultiADD ---- 18
2.2.2 - Resolução de conflitos no MultiADD ---- 20
3 - MULTIJADE ---- 22
3.1 - Modelo de interação centralizado ---- 25
4 - INFRAESTRUTURA COMPUTACIONAL DE GERENCIAMENTO E COMUNICAÇÃO DE DADOS DO AMBIENTE MULTIJADE ---- 27
4.1 - Arquitetura de servidores ---- 28
4.1.1 - Servidor de Usuários ---- 30
4.1.2 - Biblioteca de Modelos ---- 32
4.1.3 - Servidor Controlador ---- 34
4.2 - Primitivas de Serviço no MultiJADE ---- 37
5 - ESTUDO DE CASO DO AMBIENTE MULTIJADE ---- 40
5.1- Iniciando o MultiJADE ---- 40
5.2 - Criação de um modelo no MultiJADE ---- 41
5.3 - Criação de um documento ativo no MultiJADE ---- 47
5.4 - Conflito no MultiJADE ---- 48
6 - CONCLUSÃO ---- 54
7 - REFERÊNCIAS ---- 58

RESUMO

Um processo de tomada de decisão envolve a identificação de um conjunto de alternativas para a decisão, a análise destas alternativas com base em um conjunto de restrições e critérios e a seleção da alternativa que apresenta a melhor avaliação. Processos de tomada de decisão estão presentes no nosso dia a dia e, em particular, no dia a dia de organizações como empresas, corporações e instituições governamentais. Este projeto de graduação descreve a infraestrutura computacional de gerenciamento e comunicação de dados do MultiJADE, um ambiente para construção de sistemas computacionais para o apoio a tomada de decisão em processos concorrentes e distribuídos. A metodologia empregada envolveu uma ampla análise de caracterização dos processos concorrentes e distribuídos de tomada de decisão, e resultou na implementação dos mecanismos de comunicação e armazenamento de dados de um sistema de arquitetura “Centralizada”, no qual atuam um agente coordenador e agentes especialistas, sendo que todos se comunicam através de um “Servidor Controlador”.

1 - INTRODUÇÃO

Um processo de tomada de decisão envolve a identificação de um conjunto de alternativas para a decisão, a análise destas alternativas com base em um conjunto de restrições e critérios e a seleção da alternativa que apresenta a melhor avaliação. Processos de tomada de decisão estão presentes no nosso dia a dia e, em particular, no dia a dia de organizações como empresas, corporações e instituições governamentais.

A tecnologia da informação tem contribuído cada vez mais para racionalizar os processos de tomada de decisão com os quais nos deparamos no dia a dia. As contribuições variam desde a disponibilização e facilidade de acesso a bancos de dados contendo as informações importantes para o embasamento da tomada de decisão até o aumento da facilidade de comunicação entre os agentes envolvidos no processo através do uso de redes de computadores e serviços (tal como o e-mail).

Uma das maneiras mais significativas que a tecnologia de informação pode contribuir para o apoio a tomada de decisão é através da construção de sistemas computacionais autônomos que se baseiam na experiência humana para tomar eles mesmos as decisões. A investigação e estudo destes sistemas computacionais têm sido tema freqüente de pesquisa em várias áreas da Ciência da Computação, tais como, otimização e inteligência artificial.

Contudo, a construção de sistemas completamente autônomos para a tomada de decisão tem se mostrado uma tarefa muito complexa e a maioria dos resultados obtidos até o momento não tem sido aplicáveis a problemas reais.

Uma abordagem que tem sido cada vez mais investigada procura realizar uma parceria entre agentes computacionais, ditos inteligentes, e agentes humanos para a realização da tomada de decisão. Nesta abordagem, os sistemas computacionais atuam como parceiros do agente humano responsável pela decisão, realizando tarefas que vão desde a execução de cálculos matemáticos repetitivos até a sugestão de soluções, passando por tarefas de documentação automática do processo de decisão. Assim, o agente humano é quem efetivamente toma a decisão, mas esta decisão é

significativamente suportada pelos serviços prestados pelos agentes computacionais.

Nesta linha surgiu o JADE (Java based Active Documents Environment) [Varejão et al., 2001; Varejão et al., 2002, Varejão et al., 2003]. Trata-se de um sistema computacional baseado na abordagem de Documentos Ativos [Garcia, 1992] destinado a auxiliar os usuários a tomarem suas decisões e a fornecer sugestões.

Outro aspecto importante que tem sido considerado no apoio ao processo de tomada de decisão dentro da área de tecnologia da informação, é o fato de que muitas vezes o responsável pela tomada de decisão ser um grupo de pessoas e não apenas um indivíduo. Freqüentemente, estas pessoas possuem diferentes perfis de formação e trabalham em locações distantes uma das outras. Isso acaba trazendo enormes dificuldades para o processo de tomada de decisão, visto que os envolvidos no processo necessitam se comunicar efetivamente e negociar os conflitos de interesse de maneira racional. Para tanto, é necessário que cada um entenda as razões e demandas específicas dos demais. Neste caso, temos um processo concorrente e distribuído de tomada de decisão.

A tecnologia da informação pode contribuir significativamente para a redução dos problemas abordados acima fornecendo sistemas computacionais que apóiem o processo de decisão, aumentem e facilitem as possibilidades de comunicação e interação entre as partes envolvidas, registrem e forneçam acesso a todas as ações realizadas pelos diferentes participantes do processo, bem como os motivos e argumentos adotados nas tomadas de decisão. Isto pode ser feito com o uso de sistemas que registrem as decisões de maneira a poderem ser consultadas posteriormente. Desta forma o tempo que os envolvidos no problema precisariam estar juntos pode ser minimizado, uma vez que eles podem interagir também com o sistema e entender melhor as decisões tomadas pelos demais.

Uma abordagem para construção de sistemas computacionais voltados para apoiar a resolução de problemas em processos concorrentes e distribuídos com as características apontadas acima é o MultiJADE [Gama, 2004; Gama et al., 2004]. Mais especificamente são utilizadas as tecnologias

de documentos ativos de design e sistemas multiagentes, sendo que é utilizado o princípio de que os sistemas computacionais agem em parceria com os seres humanos. Este sistema permite que as decisões e as razões das decisões dos agentes sejam registradas em documentos ativos de maneira a poderem ser consultadas posteriormente.

Este projeto de graduação tem por objetivo a apresentação da infraestrutura computacional de gerenciamento e comunicação de dados do ambiente MultiJADE, sendo estruturado em mais cinco capítulos. No segundo, são discutidos os conceitos de documentos ativos de design e d*esign* concorrente e distribuído. Estes conceitos são importantes uma vez que vão servir como base para o entendimento do MultiJADE.

No terceiro capítulo são apresentados a estrutura conceitual do MultiJADE e seu funcionamento lógico. No quarto capítulo, é dado foco ao que foi realizado neste projeto de graduação, ou seja, é discutida a implementação da infraestrutura computacional de armazenamento de dados e comunicação entre agentes no ambiente MultiJADE.

Com o intuito de mostrar o uso do MultiJADE e seu funcionamento no nível de mensagens, no quinto capítulo é apresentada uma aplicação. Trata-se do projeto simplificado de um veículo automotivo, no qual vários grupos de planejamento estrutural estão envolvidos. É descrita a rede paramétrica desta aplicação bem como uma situação de conflito.

Finalmente, na conclusão, são apresentadas e discutidas as contribuições deste trabalho e sugestões para próximos passos relacionados ao MultiJADE.

2 - DOCUMENTOS ATIVOS DE DESIGN

O desenvolvimento de um projeto contempla vários processos de tomada de decisão. Estes processos exigem que além do levantamento das alternativas, seja feita uma análise de suas vantagens e desvantagens, de modo que a escolha possa ser a mais adequada.

No dia a dia das empresas nota-se que grande parte dos projetos possui apenas o registro da solução final, não sendo documentada a tomada de decisão em si, ou seja, o motivo pelo qual uma determinada opção foi escolhida (a este tipo de conhecimento chamamos *design rationale* (DR) [Lee, 1998]).

Na grande maioria das vezes, esta documentação é extremamente útil em projetos de médio e grande porte visto que possibilita uma visão mais global do projeto por parte das equipes que o integram e, conseqüentemente, um maior entendimento dos passos e motivações envolvidos no processo de tomada de decisão, facilitando assim a comunicação e negociação entre equipes. Daí é importante que este conhecimento despendido durante o projeto seja armazenado de forma amigável, possibilitando que seja facilmente recuperado quando necessário.

Uma abordagem dentro da linha de *design rationale* é a de Documentos Ativos (também conhecidos como sistemas ADD) [Garcia, 1992]. Sistemas ADD são Sistemas Baseados em Conhecimento que usam a metáfora da existência de um aprendiz computacional que acompanha o responsável humano na resolução do problema, apoiando-o e registrando suas decisões. A interação entre o ser humano e o aprendiz ocorre de duas maneiras: o ser humano pode pedir ao aprendiz que realize cálculos e tome decisões ou pode, ele mesmo, realizar essas tarefas e requerer que o aprendiz as acompanhe.

Para cada decisão tomada, o sistema compara a decisão do projetista com a decisão que seria tomada segundo o conhecimento armazenado. Se houver conflitos, o sistema sugere a decisão segundo o *rationale* adquirido, sendo que a decisão final é do usuário que pode ajustar o modelo do documento ativo. Sistemas ADD já se mostraram ferramentas valiosas no

apoio e na documentação de processos de tomada de decisão [Garcia et al., 1997].

Documentos Ativos de Design se baseiam na idéia de capturar o *rationale* por meio da transformação dos documentos de *design* de repositórios estáticos de dados em modelos computacionais do próprio *design*. Assim, em vez de o projetista registrar diretamente os dados relativos ao *rationale* na medida em que o *design* transcorre, ele apenas interage com um modelo computacional do processo de *design* e o conhecimento é registrado automaticamente no *rationale*, diminuindo a ocorrência de interrupções no processo de *design*, originadas pelo tempo despendido na documentação.

Para que o sistema ADD possa ser utilizado, é necessário que ele incorpore um modelo computacional do processo de *design* do artefato a ser produzido. O modelo computacional de *design* é representado por meio de uma rede de dependência paramétrica e os parâmetros dessa rede podem ser interligados. Neste caso, eles são associados aos parâmetros dos quais dependem para serem definidos e detêm conhecimento de como estes parâmetros os influenciam.

Os parâmetros da rede de dependência são subdivididos em três classes: primitivos, derivados e decididos [Varejão et al., 1996]. Parâmetros primitivos são aqueles que só podem ser definidos pelo usuário ou pelo contexto de uso do sistema. Parâmetros derivados podem ser definidos pelo usuário ou podem ser calculados diretamente (por meio da aplicação de uma fórmula ou algoritmo exato) a partir dos parâmetros que os influenciam. Por fim, parâmetros decididos são aqueles que modelam o processo de decisão do projetista.

Os parâmetros decididos também podem ser definidos pelo usuário ou podem ser calculados a partir dos parâmetros dos quais dependem. O que os diferencia dos parâmetros derivados é o processo de cálculo envolvido. Nos parâmetros decididos, cada parâmetro tem associado a si um conjunto de alternativas, critérios, restrições e funções de avaliação dessas alternativas a partir dos critérios e restrições. O valor calculado do parâmetro decidido é a alternativa que melhor satisfaz aos critérios e restrições existentes.

O sistema ADD provê serviços de gerenciamento das dependências presentes em suas redes paramétricas. Daí é possível o sistema explicar como chegou ao resultado de uma avaliação de projeto, avaliar o impacto da escolha de valores de alguns parâmetros sobre a avaliação dos demais e avaliar o impacto da atribuição de diferentes pesos de critérios sobre as decisões.

Em geral Interfaces de Explicação em Sistemas ADD disponibilizam um conjunto de perguntas possíveis de serem respondidas, com as quais os usuários interagem. Ele escolhe a pergunta que deseja fazer e o sistema apresenta a resposta gerada pelo módulo de explicação. De acordo com Braun [Braun et al., 2003], existe um conjunto de quatro perguntas típicas que são abordadas nos módulos de explicação de Sistemas ADD. A TABELA 1 detalha quais são estas perguntas.

TABELA 1 - Perguntas do módulo de explicação de sistemas ADD

Por quê?	$\Rightarrow$	Responde o porquê do resultado apresentado.
Por que não?	$\Rightarrow$	Quando o resultado apresentado frustra as expectativas do usuário.
E se?	$\Rightarrow$	Mostra o que aconteceria se o usuário alterasse alguma decisão.
Como?	$\Rightarrow$	Resumo do histórico de todos os passos efetuados pelo usuário e pelo sistema.

Como toda abordagem, sistemas ADD possuem vantagens e desvantagens. Por um lado, sua forma de representação do conhecimento é bastante simples, e isto facilita a aprendizagem e o entendimento. Contudo, existe um problema associado ao fato de que sistemas ADD precisam ser construídos do zero, ou seja, não há aproveitamento das partes comuns e isto torna a construção de sistemas ADD demorada e de alto custo.

2.1 - JADE

JADE (*Java based Active Documents Environment*) é um ambiente computacional que torna mais fácil a construção de sistemas ADD já que aumenta os tipos de interação entre os seres humanos e o sistema computacional. JADE torna o ambiente computacional ADD mais fácil de ser

customizado e estendido [Varejão et al., 2001] e permite que sejam reutilizados elementos comuns em diferentes sistemas.

Como um sistema orientado a objetos, JADE é baseado num *framework* de classes que possibilitam descrever cada tipo de parâmetro (primitivo, derivado e decidido) da rede de dependência. A FIGURA 1 mostra a versão corrente do *framework* em notação UML.

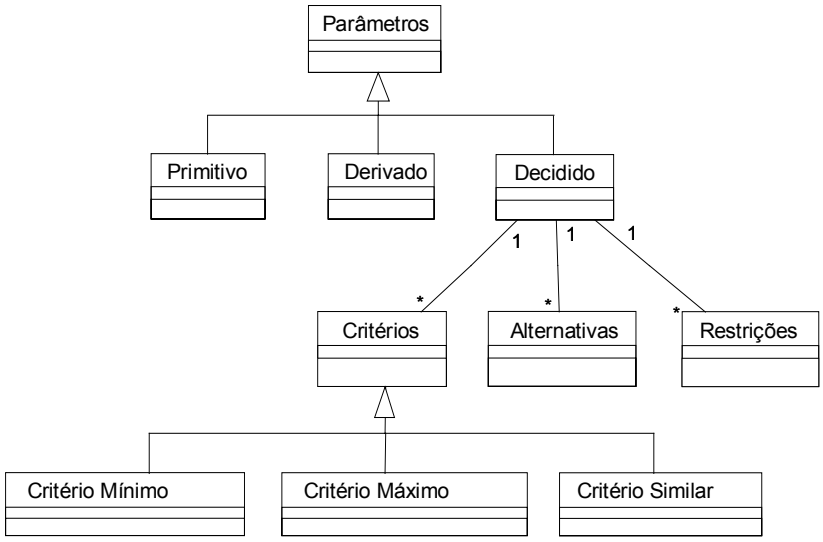

FIGURA 1 - *Framework* OO usado por JADE

Conforme mencionado, sistemas ADD usam uma rede de dependência paramétrica para modelar processos de *design*. Considerando que o conjunto primitivo de conceitos é pequeno e o fato de que a base de conhecimento pode ser visualizada graficamente, os usuários finais podem, facilmente, entender e interagir com esse sistema. Com relação a esses aspectos, sistemas ADD são mais apropriados para suportar problemas baseados em interação. JADE amplia a capacidade de interação dos sistemas ADD, permitindo o usuário final modificar mais facilmente os parâmetros e a topologia da rede de parâmetros.

JADE ameniza o esforço e o custo requerido para construir e mudar sistemas ADD, uma vez que o usuário precisa somente usar a interface de aquisição para criar e modificar a rede de parâmetros. Nesta interface são criados parâmetros, definidos seu tipo, unidade e valor padrão. Na FIGURA 2 temos uma rede simples onde A e B são parâmetros primitivos e C é um parâmetro derivado, cuja fórmula é A - B.

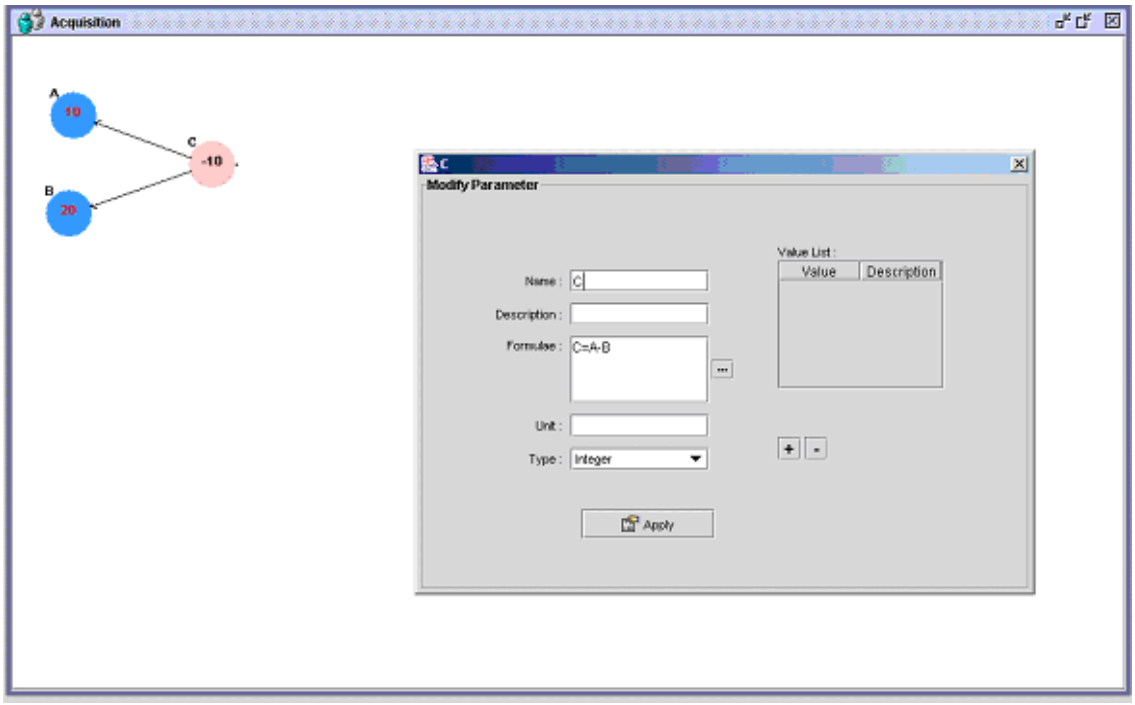

FIGURA 2 - Interface gráfica de aquisição do JADE

As interfaces de *design* e explicação são automaticamente criadas e modificadas pelo próprio ambiente JADE. A FIGURA 3 exibe a interface de explicação do JADE, na qual pode-se reparar que existem quatro janelas: Por quê?, Por que não? E se? Como?. Trata-se das quatro perguntas típicas dos módulos de explicação de sistemas ADD.

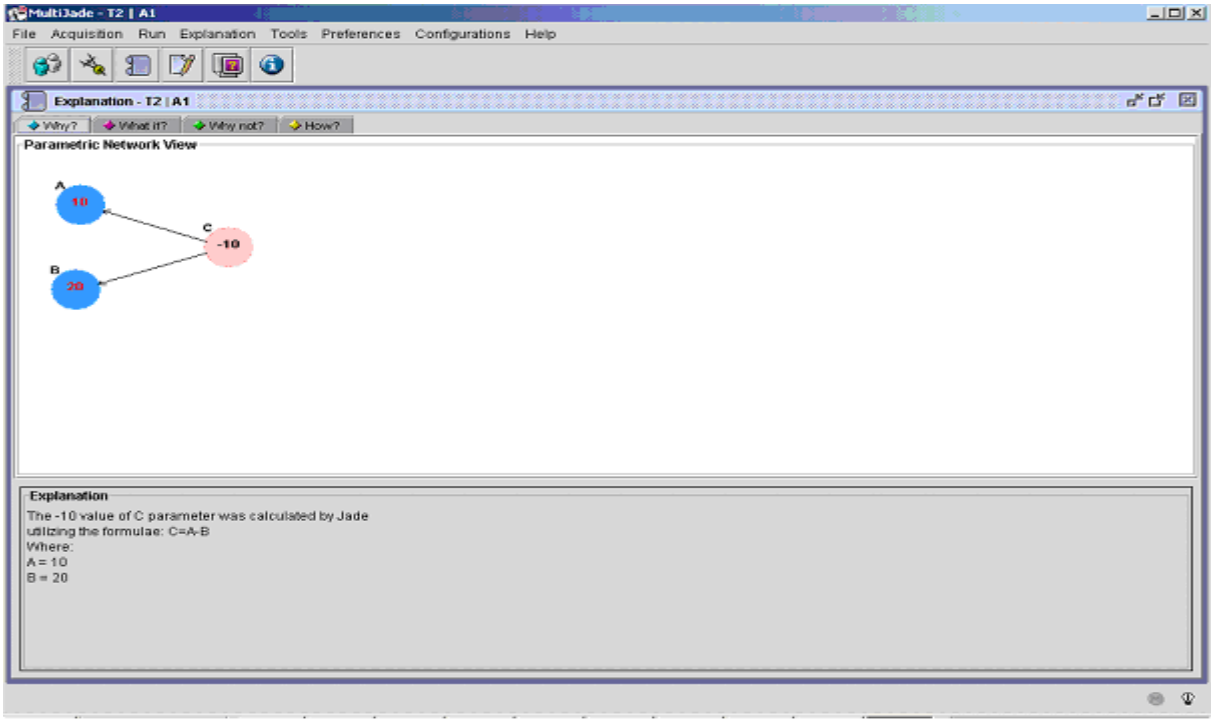

FIGURA 3 - Interface de Explicação do JADE.

Embora originalmente destinado a tarefas de *design*, JADE também pode ser aplicado a outros tipos de problema de decisão, desde que a representação por redes paramétricas seja possível.

No JADE, o *rationale* é armazenado de duas maneiras: na forma de um arquivo de log (histórico - documentação automática dos passos efetuados durante o *design* – arquivo texto) e na forma do modelo do processo de *design* (representado pela rede paramétrica e pelo estado dos parâmetros – arquivo serializado). O histórico é gerado através da documentação de todos os passos do usuário e da conseqüente resposta do JADE.

Segundo Braun [Braun et al., 2003], outro ponto importante a ser ressaltado é que no JADE existem ainda três conceitos: o modelo, o documento ativo e o estado. Modelo é um protótipo de Documento Ativo que serve para dar forma a um problema de *design*, a fim de resolvê-lo, e que vai sendo melhorado à medida que a base de conhecimento enriquece. Documento ativo herda de um modelo todos os seus atributos e pode ser especializado em um subproblema. Estado corresponde a uma configuração dos valores dos parâmetros do documento ativo.

Até agora, foram apresentadas possibilidades de uso de sistemas ADD que permitem uma maior interação e compreensão entre as partes envolvidas num processo de *design*. Todavia, isso foi feito sem levar em conta a possibilidade de vários sistemas ADD interagirem entre si para resolver problemas e conflitos. Este será o foco da próxima seção.

2.2 - Design concorrente e rationale

De acordo com Smith [Smith, 1995], a maior parte das tarefas de *design* envolve o gerenciamento de conflitos que surgem quando requisitos contraditórios são apresentados no processo de criação de um artefato. A detecção e resolução de conflitos são tarefas mais árduas quando o *design* e o conhecimento envolvido estão distribuídos entre diferentes pessoas com diferentes perspectivas.

A maior parte dos processos de *design* envolve processos sociais, nos quais projetistas interagem uns com os outros, com patrocinadores do projeto, com usuários e com outros participantes do processo, de forma a identificar necessidades, entender o problema, especificar e definir a solução e comunicar aos envolvidos como realizar a solução.

Para Finger et al. [Finger et al., 1995], a essência do design concorrente é o conjunto de interações que ocorrem na interface entre os membros da equipe de *design* e suas ferramentas. Barreiras para o processo de *design* concorrente incluem problemas de comunicação, dificuldades de tradução e perda de histórico de dados.

No *design* concorrente e distribuído, o *design rationale* pode ser usado na perspectiva automatizada ou de apoio ao usuário. Na projeção de conflitos, o *design rationale* pode manter o registro do que foi feito no projeto e por qual razão essas decisões foram tomadas. Ciente das razões que levaram as decisões tomadas no projeto, os usuários podem evitar a ocorrência de conflitos. Na resolução de conflitos efetivamente existentes, o DR pode identificar a ocorrência de conflitos e apresentar aos usuários as razões pelas quais elas ocorreram, facilitando sua mitigação.

2.2.1 - MultiADD

Sistemas ADD podem ser usados para atender às necessidades estruturais de desenvolvimento de sistemas concorrentes e distribuídos de apoio à decisão. A FIGURA 4 apresenta uma proposta para a arquitetura do MultiADD [Garcia et al., 1997].

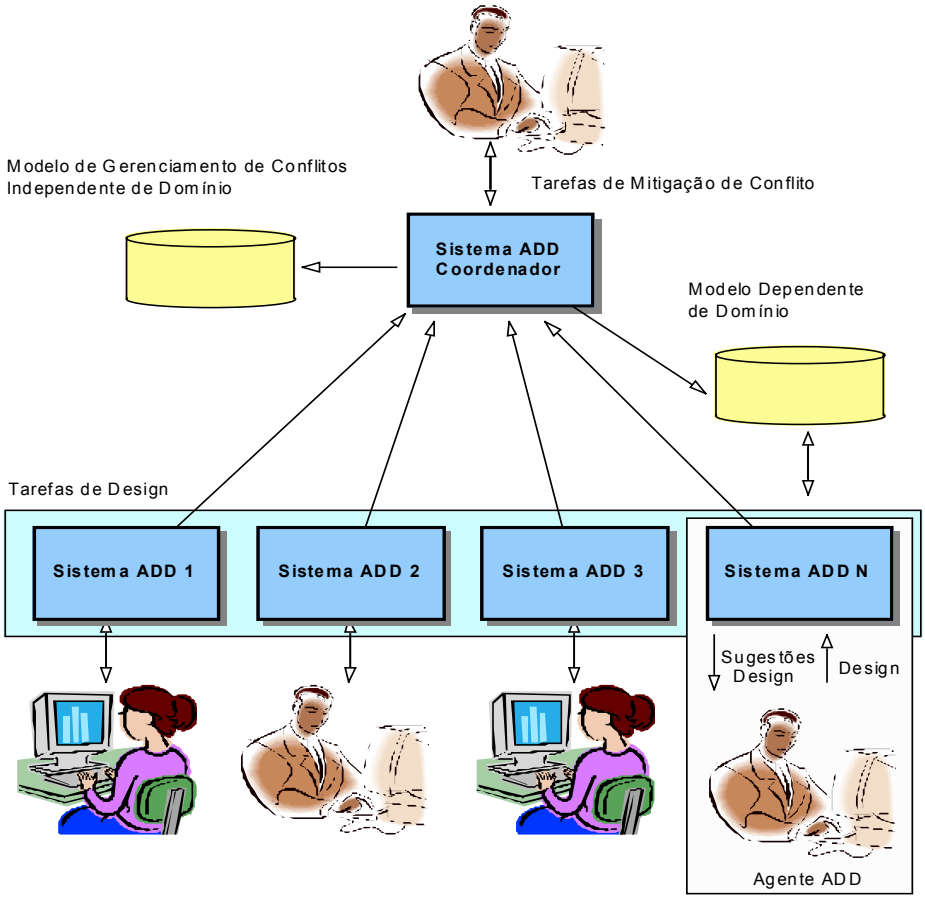

FIGURA 4 - Arquitetura multiagente para sistemas ADD

Na TABELA 2, detalhamos os papéis vistos na FIGURA 3.

TABELA 2 - Entidades da arquitetura MultiADD

Entidades	Agentes ADD	Descrição
Usuário	Parte humana do agente ADD.	Pessoa ou grupo de pessoas responsável por decisões de parte ou de todo o projeto.
Coordenador	Parte humana do agente ADD coordenador responsável pela resolução de impasses no projeto.	Pessoa ou grupo encarregado de solucionar conflitos.
Sistema ADD	Parte computacional do agente ADD.	Modelo computacional que auxilia o usuário a desenvolver uma parte do projeto.
Sistema ADD Coordenador	Parte computacional do agente ADD coordenador, que auxilia na resolução de conflitos.	Modelo computacional que auxilia o usuário a coordenar o projeto e resolver conflitos.

De acordo com Garcia et al. [Garcia et al., 1997], os agentes ADD permitem aos usuários criar um *design* auto-explicativo. Assim, outros usuários

e projetistas podem ter acesso a explicações a respeito de todas as decisões do *design*. O MultiADD aumenta a sinergia do grupo e reduz as horas de reunião necessárias para explicação do projeto, permitindo que os projetistas explorem informações sobre o projeto e *rationale* de outras partes de um mesmo projeto ou até de projetos antigos dentro de um mesmo escopo.

A estratégia do MultiADD para tratar conflitos inclui a identificação e resolução dos impasses por um controlador central. Na próxima seção, será abordada a forma de resolução de conflitos no MultiADD.

2.2.2 - Resolução de conflitos no MultiADD

Assim como o ADD, o MultiADD utiliza uma rede de parâmetros para coletar informações do projeto. Conforme Garcia et al. [Garcia et al., 1997], em um *design* parametrizado, um conflito ocorre quando decisões de um projetista afetam decisões de outro projetista. Um conflito representa a existência de diferentes valores atribuídos por agentes diferentes a um mesmo parâmetro.

Numa estrutura MultiADD, em que existem as figuras dos projetistas e de um controlador, este deve estar informado a respeito de todos os parâmetros que estejam em conflito. Garcia et al. [Garcia et al., 1997] propõem duas formas para que o controlador possa estar ciente das informações de conflito:

- Aguardar que a equipe de *design* informe ao controlador sobre o conflito;
- O controlador verifica periodicamente as variáveis de conflito com o objetivo de averiguar se ocorreu um conflito.

De qualquer forma, assim que o conflito for detectado, uma ação deve ser tomada. Comparada com a tarefa de “o que fazer com o conflito?”, a tarefa de identificar o conflito é considerada fácil. Em alguns casos, é necessário um conjunto de ações para resolver um único conflito. Garcia et al. [Garcia et al., 1997] propõem algumas ações a serem executadas mediante ocorrências de conflito, através do enquadramento destes conflitos em algumas pré-condições. Basicamente as pré-condições checam se todos os agentes foram envolvidos ou apenas parte dos mesmos, e persistência do conflito (se vem ocorrendo

com freqüência, se não houve consenso na resolução do mesmo). Uma vez verificadas as pré-condições, são tomadas as ações que consistem em avisar aos agentes envolvidos ou ao coordenador, conforme pré-condição identificada.

3 - MULTIJADE

Conforme visto no capítulo anterior, o ambiente computacional JADE não trata problemas concorrentes e distribuídos. Como esta abordagem tem mostrado ser bastante interessante na construção de Sistemas ADD, surge uma proposta de extensão e adaptação do JADE, permitindo que sejam suportados diferentes tipos de interações e estratégias de resolução de conflitos em uma sociedade de agentes: o MultiJADE [Gama, 2004; Gama et al., 2004].

Podemos entender a arquitetura lógica do sistema MultiJADE utilizando a metáfora de uma “mesa virtual” (vide FIGURA 5), na qual agentes informam suas decisões e negociam eventuais conflitos, sempre em busca de uma solução que seja a melhor para todo o projeto.

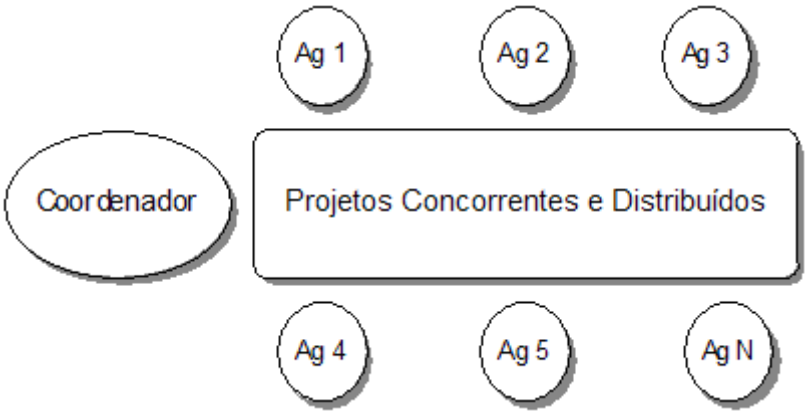

FIGURA 5 - Mesa virtual de negociação no MultiJADE

Uma vez que se trata de uma reunião virtual, não há a necessidade de todos os participantes estarem presentes ao mesmo tempo. Cada um deles pode realizar sua atividade em separado nos momentos em que se encontram no sistema.

Parâmetros de comunicação são parâmetros de interface das redes paramétricas dos agentes do modelo e, o relacionamento entre parâmetros dos agentes é definido através de regras agregadas pelo engenheiro de conhecimento. Caso essas regras não sejam satisfeitas é gerado um conflito. Os parâmetros de comunicação são sempre utilizados por dois ou mais agentes, conforme apresentado na FIGURA 6.

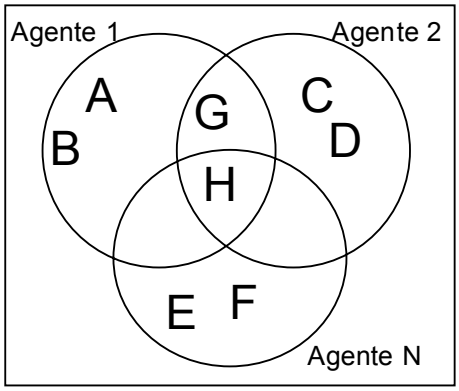

FIGURA 6 - Parâmetros de Comunicação no MultiJADE

O parâmetro “H” é definido como parâmetro de comunicação entre os agentes 1, 2 e N e o parâmetro “G” é definido como parâmetro de comunicação entre os agentes 1 e 2. Os demais parâmetros não são considerados parâmetros de comunicação. Sendo assim, A e B são parâmetros específicos do Agente 1, C e D do Agente 2 e E e F do Agente N.

As relações dos parâmetros entre agentes são definidas dois tipos: relações de igualdade e relações gerais. As relações de igualdade definem regras entre os parâmetros dos agentes que exigem que os valores dos parâmetros sejam iguais, conforme representado na EQUAÇÂO (1):

$$\text{Parâmetro A1} = \text{Parâmetro A2} = \text{Parâmetro A3} \qquad (1)$$

As relações gerais ou abertas são todas as relações entre os parâmetros dos agentes que não são de igualdade, ou seja, elas admitem fórmulas e expressões entre os parâmetros. A EQUAÇÂO (2) ilustra um exemplo de relação aberta:

$$\text{Parâmetro A1} > (\text{Parâmetro A2} + \text{Parâmetro A3}) \qquad (2)$$

Existem dois tipos distintos de relação (igualdade e gerais) devido à ação que deve ser tomada quando um determinado parâmetro é alterado. Na relação de igualdade sempre que um parâmetro é alterado por um agente, os demais agentes envolvidos na relação são avisados e podem aceitar ou recusar uma sugestão do valor do parâmetro. Um conflito só é gerado quando um dos agentes recusa este valor. Já na relação aberta, os agentes envolvidos só tomam conhecimento de que há uma situação na qual as regras não estão sendo satisfeitas e que por isso foi gerado um conflito.

Quando ocorre um conflito, os agentes envolvidos têm um tempo pré-estabelecido para interagirem e chegarem a uma solução. A este período denominamos “Tempo de Mitigação do Conflito” e é o agente coordenador o responsável por sua configuração de acordo com o impacto que o conflito pode ter sobre o bom andamento do projeto.

No MultiJADE, uma vez que um conflito é resolvido dentro de seu tempo de mitigação, volta-se ao estado de equilíbrio. Entretanto, se o tempo de mitigação cessar e o conflito não tiver sido resolvido, o agente coordenador deve interferir no processo de negociação com o intuito de evitar que o problema se prolongue por um grande período de tempo, o que seria prejudicial para o andamento do projeto como um todo. Após a intervenção do coordenador para resolução do impasse, tem-se restabelecido o estado de equilíbrio. A FIGURA 7 ilustra o fluxo compreendido entre a identificação de um conflito e a solução do mesmo.

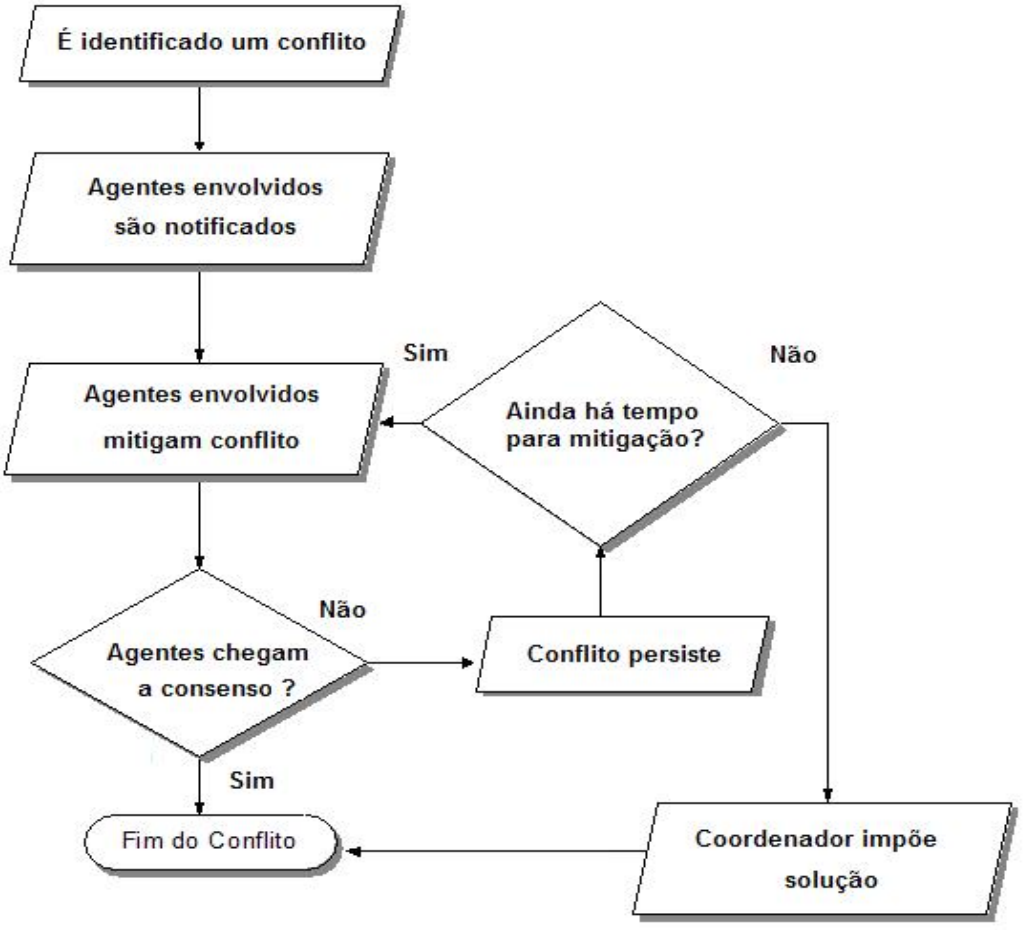

FIGURA 7 - Fluxograma de mitigação de um conflito

O MultiJADE auxilia projetos distribuídos uma vez que disponibiliza uma estrutura para registro de decisões e recuperação das mesmas. Desta forma, mesmo que num determinado momento não haja disponibilidade de todos os envolvidos no projeto para debate de um determinado problema, fica assegurado que todos os envolvidos terão acesso às decisões tomadas e às justificativas das mesmas.

É importante frisar que a proposta do MultiJADE não é propriamente de uma ferramenta de resolução de conflitos, mas sim de um sistema que possa identificar e dar suporte a mitigação de conflitos de forma que os usuários possam chegar a uma solução de maneira mais eficiente.

3.1 - Modelo de interação centralizado

Ao longo do projeto da arquitetura computacional do ambiente MultiJADE, percebemos que seria possível a utilização de vários modelos de interação entre agentes. Em virtude do grande porte e complexidade de cada um desses modelos, optamos por implementar apenas o modelo de interação centralizado, deixando os demais para projetos futuros.

Neste modelo de interação existem as figuras dos agentes específicos, responsáveis pela tomada de decisão, e do Controlador, responsável pela comunicação entre agentes, identificação de conflitos e suporte a resolução destes. Os agentes específicos não se comunicam diretamente entre si, eles enviam mensagens para o Controlador que repassa as informações convenientes para os demais agentes na ordem e momento apropriado. A FIGURA 8 ilustra este modelo.

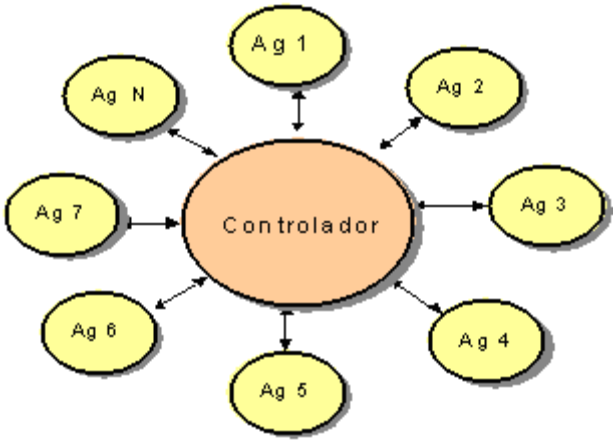

FIGURA 8 - Modelo de interação centralizado

Como veremos mais adiante, além de ser responsável pelo roteamento de mensagens entre agentes, identificação e suporte a mitigação de conflitos, o Controlador será o repositório dos documentos ativos de um projeto.

4 - INFRAESTRUTURA COMPUTACIONAL DE GERENCIAMENTO E COMUNICAÇÃO DE DADOS DO AMBIENTE MULTIJADE

No MultiJADE, os modelos servem de base para a construção dos documentos ativos utilizados em projetos. Na FIGURA 9 é apresentado um modelo criado pelo construtor coordenador e que posteriormente, é compartilhado com construtores especialistas.

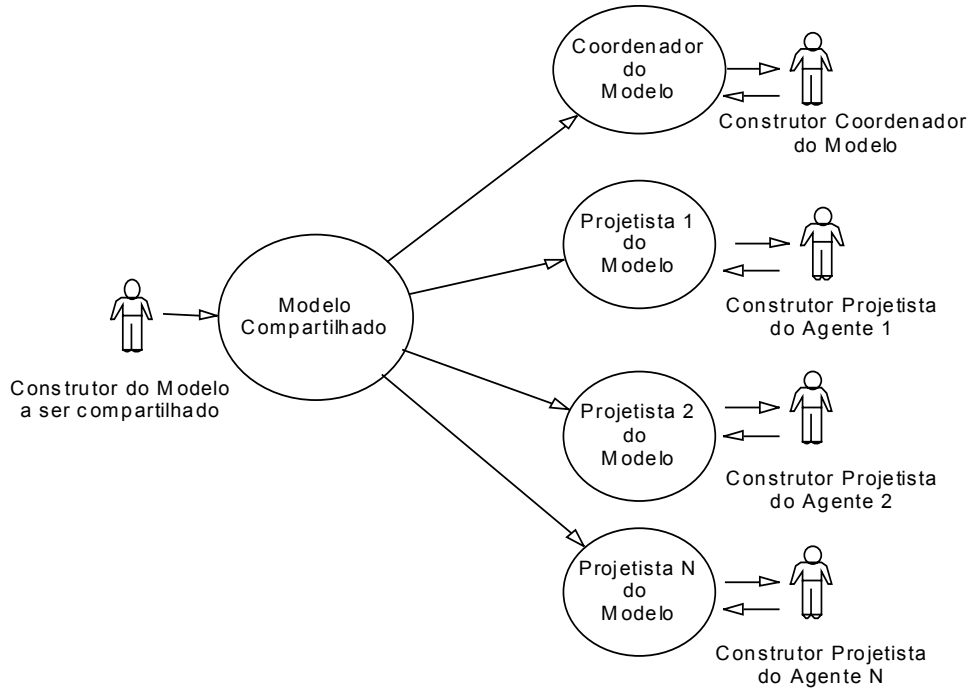

FIGURA 9 - Construção de um modelo no MultiJADE

Durante a fase de criação de um modelo o engenheiro do conhecimento especifica o número de agentes e os parâmetros que serão comuns às redes paramétricas dos agentes do modelo. Ao criar os parâmetros, o engenheiro do conhecimento define quais são os parâmetros de comunicação, as relações para estes e a quais agentes eles pertencem.

Uma vez concluído o modelo, entram em cena os engenheiros especialistas que representam cada agente do modelo. O papel dos engenheiros especialistas é complementar o modelo criado pelo engenheiro do conhecimento. Esta complementação consiste em adicionar aos seus respectivos agentes as informações mais específicas através da criação de

novos parâmetros, sendo que novos parâmetros de comunicação não podem ser mais adicionados.

Após o modelo dos agentes ter sido complementado pelos especialistas, ele é considerado concluído e pode ser disponibilizado para criação de um ou mais documentos ativos.

A criação de um documento ativo no MultiJADE é feita a partir de um modelo já existente e concluído. Conforme mencionado, um documento ativo é uma instância de um modelo, como visualizado na FIGURA 10.

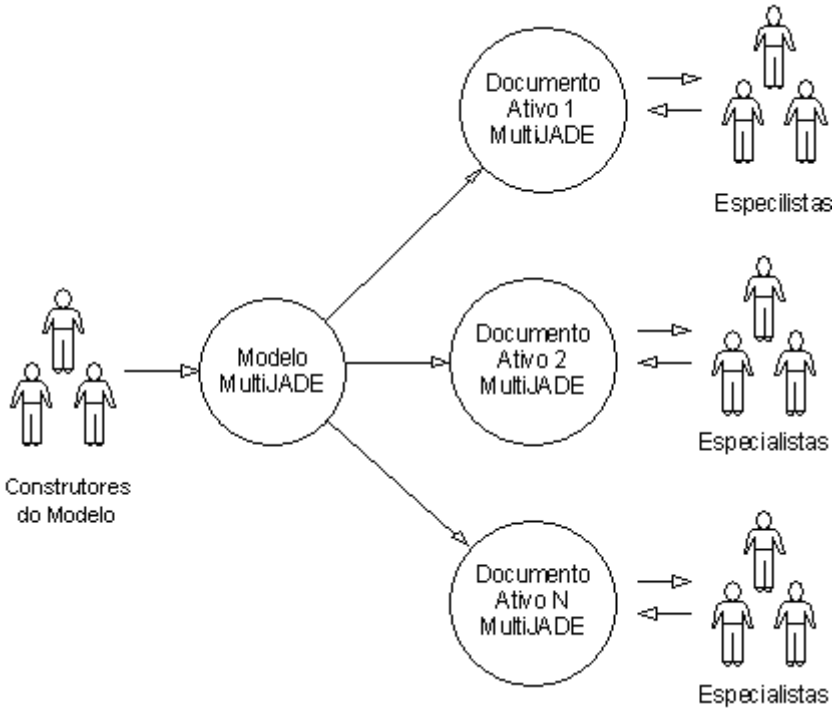

FIGURA 10 - Modelo e Documento Ativo no MultiJADE

Aqui, foi apresentada uma visão geral dos passos compreendidos na criação e complementação de um modelo no MultiJADE e, sua posterior instanciação na forma de documento ativo em um projeto. Na próxima seção serão abordados detalhes de implementação das funcionalidades do ambiente MultiJADE.

4.1 - Arquitetura de servidores

Apontadas às características desejáveis e o comportamento do sistema MultiJADE durante o processo de mitigação de conflitos, nesta seção será

apresentada a arquitetura computacional, sendo principalmente focado o processo de comunicação.

Durante a concepção do MultiJADE percebemos que além do ambiente de *design*, seriam necessárias entidades computacionais para que fossem alcançadas as características de funcionamento almejadas. Então, após estudo, foi modelada uma arquitetura computacional que é composta por um Servidor de Usuários, um Servidor de Modelos, um Servidor Controlador e os Agentes MultiJADE [Gama, 2004]. Na FIGURA 11 é apresentada esta arquitetura.

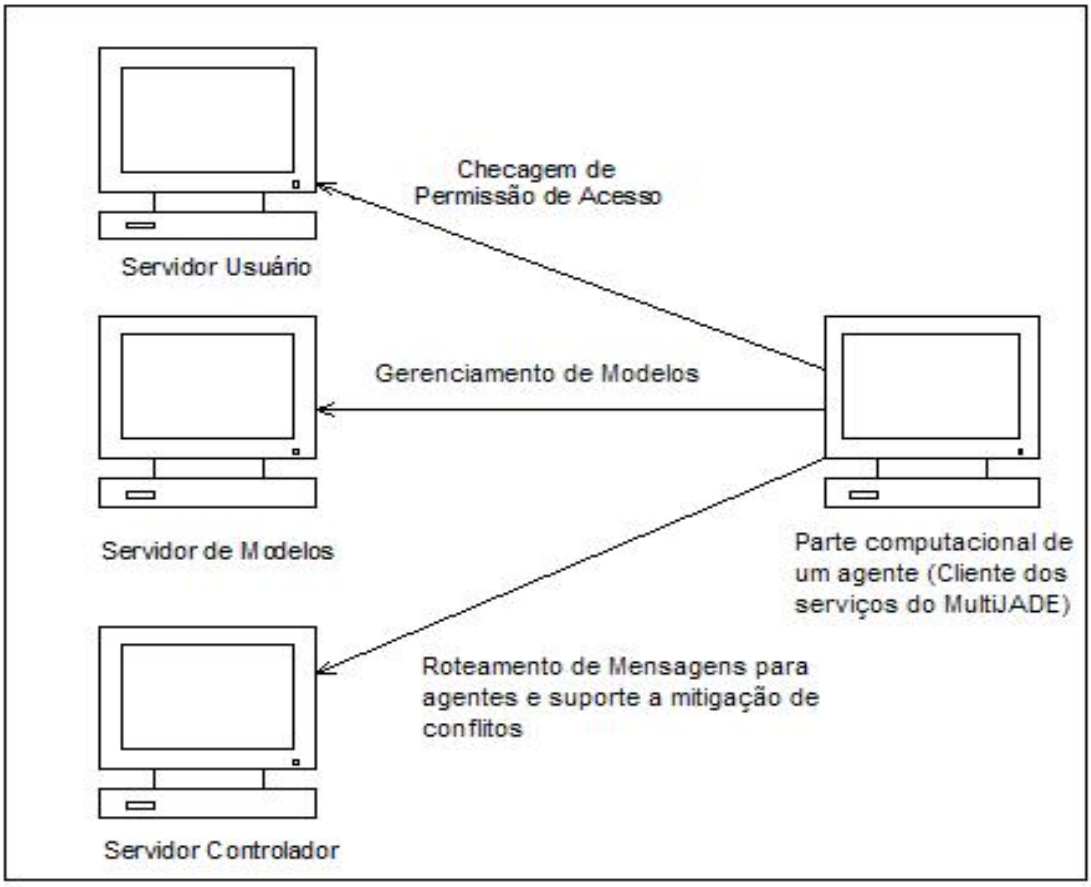

FIGURA 11 - Arquitetura computacional do MultiJADE

Definida a arquitetura do sistema MultiJADE, o passo subseqüente consistiu na sua implementação. Este projeto de graduação teve como principal foco o desenvolvimento da infraestrutura computacional de gerenciamento e comunicação de dados do sistema MultiJADE, onde foram implementados os servidores e as primitivas de comunicação entre as entidades do sistema.

A implementação do projeto foi feita na linguagem de programação Java, seguindo o mesmo padrão adotado para o ambiente JADE, o qual já vinha

sendo desenvolvido nesta mesma linguagem. Para isto, durante este projeto foi estudada esta linguagem de programação, com a qual foram implementados os módulos de software. Como parte do aprendizado, foi utilizada a implementação corrente da ferramenta computacional JADE. Contudo, o foco do estudo foi o aprofundamento dos estudos na parte de computação distribuída em Java.

Nas próximas subseções será apresentada a contribuição deste projeto de graduação, ou seja, serão apresentados detalhes de implementação dos servidores e o conjunto de primitivas compreendidas no processo de comunicação.

4.1.1 - Servidor de Usuários

Este servidor é responsável pelo controle de acesso de usuários ao sistema MultiJADE. É ele quem verifica se determinado usuário está cadastrado ou não no sistema, permitindo ou não seu acesso.

A FIGURA 12 ilustra a interface das funcionalidades do Servidor de Usuários disponíveis para seu administrador.

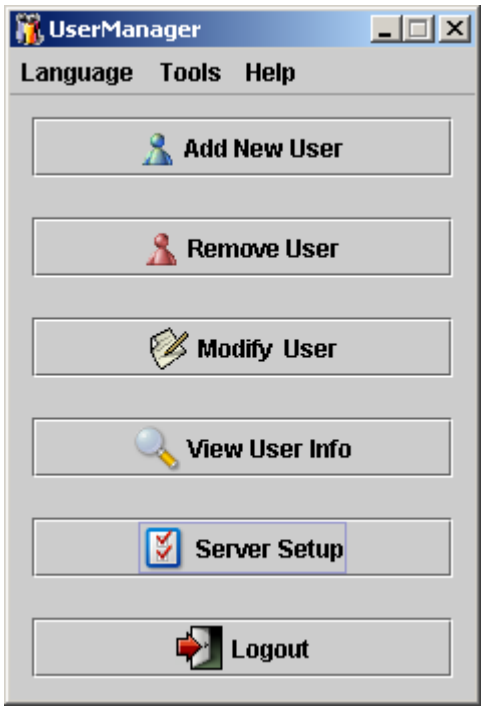

FIGURA 12 - Interface do Servidor de Usuários

Funcionalidades:

- "Add New User": Adiciona um novo usuário ao sistema;
- "Remove User": Remove um usuário do sistema, impossibilitando-o de utilizar o ambiente MultiJADE;
- "Modify User": Modifica informações do usuário, tais como senha, nome e e-mail de contato;
- "View User Info": Visualiza informações dos usuários;
- "Server Setup": Possibilita a configuração da porta de trabalho e inicia monitoramento de requisições;
- "Logout": Fecha o Servidor de Usuários, suspendendo todos os seus serviços.

Esta interface foi desenvolvida durante este projeto de graduação e foi elaborada depois de uma análise de quais funcionalidades seriam desejáveis.

Além das funcionalidades apresentadas, o Servidor de Usuários possui ainda um cliente de e-mail que pode ser utilizado no suporte a usuários do sistema MultiJADE. A implementação utilizada foi a *Yet Another Mail Manager* - YAMM [Ehnbom, 2000]. Foi necessária a customização deste módulo de software para que fossem atendidas as necessidades do sistema MultiJADE, e isto demandou um grande esforço, pois foi necessário analisar o código fonte desta implementação de cliente de e-mail.

A implementação do Servidor de Usuários foi realizada da seguinte forma: ao ser configurado e iniciado através do botão “Server Setup”, é criada uma instância da classe UserServer, que tem o papel de monitorar requisições de comunicação que chegam a uma determinada porta de comunicação do protocolo TCP/IP. Esta instância permanece ativa até que o Servidor de Usuários seja desligado.

Quando uma requisição de comunicação é recebida, é criada uma instância da classe UserReceiver para tratá-la e o servidor volta a monitorar a porta de comunicação como especificado anteriormente, com o intuito de receber novas requisições. A instância da classe UserReceiver é uma thread que continua ativa durante todo o processo de comunicação entre o servidor e o cliente, a não ser que ocorra alguma falha, e neste caso, o processo de comunicação deve ser reiniciado.

As requisições que podem ser feitas pelo cliente ao Servidor de Usuários são: “Cadastrar Usuário”, “Remover Usuário”, “Modificar Informações de Usuário”, “Autenticar Usuário” e “Obter lista de Usuários”. Na seção de primitivas de serviço serão mostrados detalhes destas funcionalidades.

Deve-se ter em mente que como se trata de um servidor concorrente, as operações mencionadas possuem um mecanismo de sincronização que se faz presente na classe UserList, a qual possui a palavra reservada *synchronized* à frente de seus principais métodos.

A persistência dos dados, ou seja, dos usuários existentes neste servidor é feita através do armazenamento de um arquivo serializado. Na tentativa de evitar a perda de informações por falha ou interrupção indevida do Servidor de Usuários, sempre que uma operação de modificação da lista de

usuários ou de dados de um usuário é realizada, é atualizado o arquivo de armazenamento.

É importante ressaltar ainda que para se executar o Servidor de Usuários, basta o administrador ir ao diretório onde suas classes estão e, chamar via console a classe UserManager (java UserManager).

4.1.2 - Biblioteca de Modelos

Como vimos anteriormente, modelos são construídos para serem utilizados posteriormente na criação de documentos ativos. Surge então a necessidade de um repositório ou biblioteca para que estes modelos sejam armazenados.

A Biblioteca de Modelos ou Servidor de Modelos surge com este intuito e dá todo o suporte tanto na fase de criação quanto na utilização dos modelos para construção de documentos ativos.

Podemos dividir os modelos em dois grupos: modelos em criação, que podem ser editados normalmente a qualquer momento, e modelos criados, que não podem mais ser alterados. Somente modelos criados geram documentos ativos que são utilizados em projetos.

Como funcionalidades da interface do Servidor de Modelos disponíveis para seu administrador podemos citar “Visualizar Modelos”, “Remover Modelos” e “Configurar Servidor”. A interface do Servidor de Modelos desenvolvida durante este projeto de graduação foi elaborada depois de uma análise de quais funcionalidades seriam desejáveis para seu administrador. A FIGURA 13 ilustra esta interface.

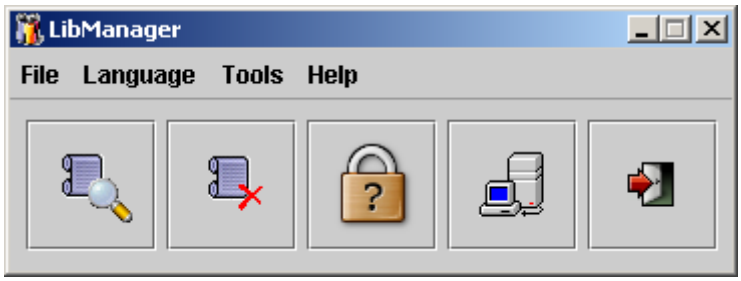

FIGURA 13 - Interface do Servidor de Modelos

Assim como o Servidor de Usuários, o Servidor de Modelos utiliza uma customização da implementação de cliente de e-mail [Ehnbom, 2000] que foi

feita durante este projeto. Este cliente de e-mail pode ser utilizado no suporte a usuários do sistema MultiJADE.

Após a configuração e inicio do Servidor de Modelos, é criada uma instância da classe LibServer que tem o papel de monitorar requisições de comunicação que chegam a uma determinada porta. No momento que uma requisição chega, é criada uma instância da classe LibReceiver (thread) para tratá-la e o servidor volta a monitorar a porta de comunicação para receber novas requisições.

Dentre as requisições que podem ser feitas por um agente cliente ao Servidor de Modelos podemos destacar: “Obter Lista de Modelos”, “Fazer download de Modelo”, “Fazer download de Agente do Modelo”, “Fazer upload de Modelo”, “Fazer upload de Agente do Modelo” e “Concluir Modelo”. Detalhes do funcionamento destas requisições serão dados na seção de primitivas de serviço.

Este servidor possui um mecanismo de sincronização que se faz presente principalmente na classe DirManager. Esta classe é responsável por todo o acesso a disco feito pelo Servidor de Modelos. O mecanismo de sincronização utilizado permite o acesso simultâneo e coordenado de vários usuários que tentam ler e escrever seus modelos concorrentemente. O mecanismo utilizado foi uma implementação de monitores chamada “Take-a-Number Monitor” [Christopher et al., 2001], na qual cada escritor ou leitor que chega, recebe um número e, o atendimento é realizado de acordo com a seqüência destes números, ou seja, pela ordem de chegada. No entanto, todos os leitores que possuem números consecutivos são servidos ao mesmo tempo, otimizando assim o processo de leitura que pode ser realizado simultaneamente. Esta implementação é encontrada na classe ReadersWriters.

Neste servidor, a persistência dos modelos tanto na fase de criação quanto na fase de criados é feita através do armazenamento de arquivos serializado e texto. Os arquivos serializados representam a estrutura do modelo propriamente dito, ou seja, possuem toda a rede paramétrica e demais informações relativas ao modelo. Os arquivos texto armazenam as informações de log que são geradas durante a manipulação do modelo pelo usuário.

Para se executar o Servidor de Modelos, o administrador deve ir ao diretório de instalação e chamar via console a classe LibManager (java LibManager).

4.1.3 - Servidor Controlador

Como visto no capítulo anterior, no modelo de interação centralizado existe um controlador, responsável pela comunicação, e agentes específicos, responsáveis pela tomada de decisão no problema. Os agentes específicos não se comunicam diretamente entre si. Eles enviam mensagens para o controlador que repassa as informações necessárias para os demais agentes na ordem e momento apropriado.

O Servidor Controlador implementa as características do controlador do modelo de interação centralizado, dando todo suporte durante o desenvolvimento de um projeto. Além disso, é ele quem gerencia os documentos ativos de um projeto.

Um conflito é identificado pelo Servidor Controlador quando uma condição para um dado parâmetro não é satisfeita. Neste caso todos os agentes envolvidos no conflito são comunicados e devem interagir de forma a resolver o conflito. Esta interação consiste em tentar entender os motivos pelo qual o conflito foi gerado. Para isto, o Servidor Controlador oferece funcionalidades como uma ferramenta de C*hat* e consulta ao *rationale* de outros agentes. Com estes recursos, os agentes podem propor uma melhor solução para o conflito, uma vez que possuem conhecimento prévio das necessidades dos demais agentes.

O serviço de C*hat* utilizado foi o *Babylon* Chat [McLaughlin] que é um software *open source* que pode ser obtido gratuitamente na internet. Foi necessária a customização deste módulo para que fossem alcançadas características almejadas no sistema MultiJADE, e isto demandou um grande esforço, pois foi necessária uma ampla análise do código fonte desta implementação. Vale ressaltar que o servidor de *Chat* fica no Controlador e seus clientes nos agentes.

As principais funcionalidades do Servidor Controlador disponíveis a seu administrador são: ”Configurar e iniciar Servidor”, “Iniciar Servidor de Chat” e “Remover Projeto”. A interface do servidor foi desenvolvida após análise de quais características eram desejáveis. A FIGURA 14 ilustra a interface do Servidor Controlador.

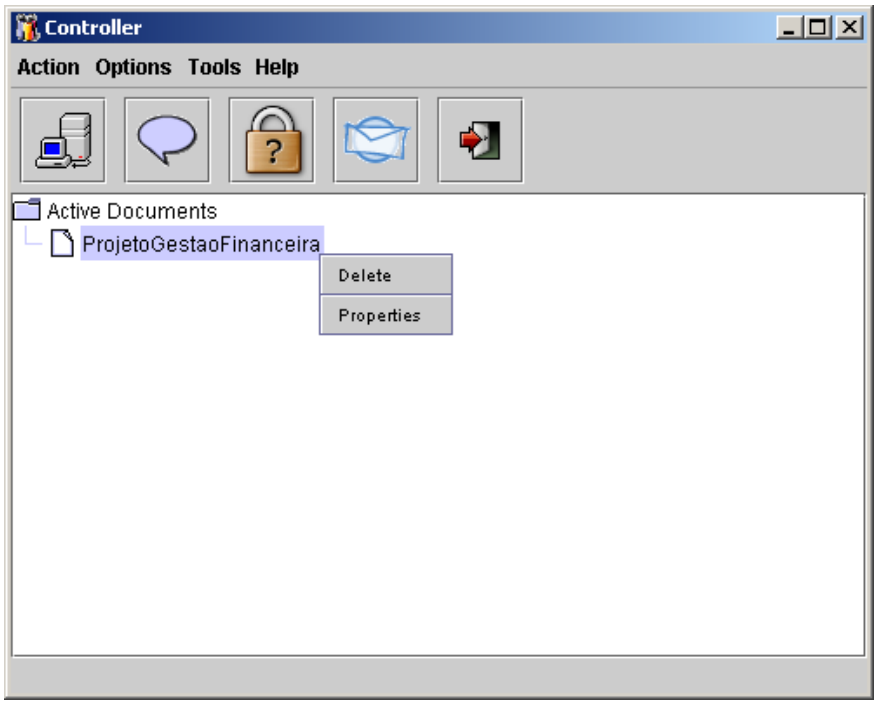

FIGURA 14 - Interface do Servidor Controlador

Assim como os demais servidores, o Servidor Controlador incorpora uma customização do cliente de e-mail YAMM [Ehnbom, 2000] que pode ser utilizado no suporte a usuários durante o desenvolvimento de um projeto no sistema MultiJADE. Este mesmo cliente de e-mail existe no ambiente de trabalho dos agentes do sistema MultiJADE.

O Servidor Controlador foi implementado para funcionar da seguinte maneira: ao configurar e iniciar o servidor é criada uma instância da classe CtrlServer que tem o papel de monitorar uma porta para receber requisições de comunicação. No momento que uma requisição de comunicação chega, é criada uma instância da classe CtrlReceiver (thread) para tratá-la e o servidor volta a monitorar a porta de comunicação para receber novas requisições.

As principais requisições que podem ser feitas ao Servidor Controlador são: “Criar Projeto”, “Fazer download de agente do projeto”, “Fazer upload de agente do projeto”, “Consultar Seções de Chat Salvas”, “Consultar e-mails salvos”, “Atualizar parâmetro de comunicação”, “Recusar atualização de

parâmetro de comunicação” (relação de igualdade), “Notificação de agente on-line” e “Notificação de agente off-line”. Na próxima seção estas e outras primitivas serão mais bem apresentadas.

Este servidor possui um mecanismo de sincronização existente em todas as classes que são acessadas concorrentemente. Para operações que demandam acesso a disco, tem-se a classe DirManager. Para operações de notificação de modificação de parâmetro, tem-se a classe ProjectManager. Assim como no Servidor de Modelos, fez-se necessário à utilização de um mecanismo de sincronização que permita o acesso simultâneo e coordenado de vários usuários que tentam ler, escrever e modificar parâmetros de comunicação em seus projetos. Da mesma forma, o mecanismo utilizado foi a implementação de monitores “Take-a-Number Monitor” [Christopher et al., 2001].

Outra característica relevante do Servidor Controlador é que nele foi incorporado o mesmo interpretador Python utilizado no ambiente JADE. Este interpretador, implementado em Java, é acionado para validar as relações dos parâmetros de comunicação assim que é feita uma modificação pelos agentes. É a partir da avaliação deste módulo de software que o Controlador toma as providências apropriadas durante o desenrolar de um projeto.

Visto que os documentos ativos herdam as características dos modelos, a persistência daqueles também é feita através do armazenamento de arquivos serializado e texto. Os arquivos serializados possuem a estrutura do documento ativo e os arquivos texto armazenam as informações de log que são geradas durante a manipulação do documento ativo pelo usuário. Todo o gerenciamento destes arquivos é feito por este servidor.

As informações de gerenciamento são armazenadas em arquivos serializados no diretório relativo a cada projeto. Exemplo disto são as mensagens que não podem ser prontamente entregues a agentes porque estes se encontram offline. Estas mensagens serão devidamente entregues no momento que o agente em questão vir a ficar online.

O administrador do Servidor Controlador pode iniciá-lo indo ao diretório de instalação e chamando via console a classe Controller (java Controller). Detalhes do fluxo de mensagens entre agente e Controlador será abordado no próximo capítulo.

4.2 - Primitivas de Serviço no MultiJADE

No MultiJADE, toda comunicação é feita através de troca de mensagens, e estas podem ter objetivos distintos. Para que um agente MultiJADE possa se comunicar adequadamente com os servidores apresentados nas seções anteriores, existe um conjunto de primitivas de serviço que se encontram na classe Messages. Na TABELA 3, são apresentadas as principais primitivas e sua descrição.

TABELA 3 - Primitivas de serviço

Primitiva	Descrição
1. addUser	Adiciona novo usuário ao sistema.
2. removeUser	Remove um usuário do sistema.
3. updateUser	Atualiza as informações cadastrais de um determinado usuário.
4. confirmUser	Autentica a senha de um usuário, deixando-o ou não entrar no sistema.
5. getUsers	Lista todos os usuários cadastrados no sistema.
6. createModel	Cria novo modelo na Biblioteca de Modelos.
7. getCreatingModels	Lista os modelos que estão em fase de criação.
8. getCreatedModels	Lista os modelos concluídos
9. getCreatingAgents	Lista os agentes de um modelo em criação.
10. getCreatedAgents	Lista os agentes de um modelo criado.
11. downloadCreatingModel	Cria uma cópia local de um modelo em criação que se encontra no Servidor de Biblioteca.
12. downloadCreatingAgent	Cria uma cópia local de um agente de um modelo em criação que se encontra no Servidor de Biblioteca.
13. uploadCreatingModel	Atualiza modelo em criação na Biblioteca de Modelos.
14. uploadCreatingAgent	Atualiza agente de um modelo em criação na Biblioteca de Modelos.
15. removeCreatingModel	Remove modelo em criação do Servidor de Biblioteca.
16. removeCreatedModel	Remove modelo criado do Servidor de Biblioteca.

17. parcialConcludeAgent	Conclui a edição de um agente de um modelo em criação.
18. parcialConcludeModel	Conclui a edição inicial de um modelo em criação, possibilitando que especialistas editem os agentes deste modelo.
19. isAgentParcialConcluded	Verifica se um determinado agente de um modelo em criação terminou de ser editado.
20. isModelParcialConcluded	Verifica se um modelo em criação terminou de ser editado.
21. finalConcludeModel	Finaliza a edição de um modelo. O status deste modelo passa de “em criação” para “criado”.
22. downloadCreatedAgent	Cria uma cópia local de um agente de um modelo que já foi concluído.
23. createProject	Cria um novo projeto no Servidor Controlador.
24. getProjects	Lista projetos existentes no Controlador.
25. getAgents	Lista agentes de um projeto no Controlador.
26. downloadAgent	Cria uma cópia local de um agente de um projeto que está no Servidor Controlador.
27. firstUpload	Armazena agente e registra seus parâmetros de comunicação no Controlador (é chamada durante a criação de um projeto).
28. uploadAgent	Atualiza o documento ativo de um agente de um projeto no Controlador.
29. removeProject	Remove um determinado projeto do Controlador.
30. getMailList	Lista os e-mails salvos de um determinado agente.
31. getChatList	Lista as seções de Chat salvas de um determinado agente.
32. downloadMail	Cria uma cópia local de uma mensagem de e-mail de um agente que se encontra no Controlador.
33. downloadChat	Cria uma cópia local de uma seção de Chat de um agente que se encontra no Controlador.
34. uploadMail	Salva uma mensagem de e-mail enviada por um agente no Controlador.
35. uploadChat	Salva uma seção de Chat de um agente no Controlador.

36. setOnline	Requisita ao Controlador permissão para ficar on-line. Somente um usuário pode ter acesso a determinado agente de um projeto em um dado instante.
37. setOffline	Agente comunica ao Controlador a intenção de se desconectar.
38. updateParameter	Atualiza um dado parâmetro de comunicação na base de dados do Controlador.
39. refuseChange	Comunica ao Controlador a recusa do valor de um parâmetro de comunicação sugerido por um outro agente. Esta primitiva é utilizada somente para parâmetros que possuem relação de igualdade.
40. imposeParameter	Após o término do tempo de mitigação cabe ao Coordenador do projeto a resolução do conflito. Com esta primitiva o Coordenador impõe um valor a um dado parâmetro de comunicação em cada agente.
41. acceptChange	Indica ao Controlador que o agente aceita a sugestão feita por outro agente para o valor de um determinado parâmetro de comunicação cuja relação é de igualdade.
42. setMitigationTime	Permite ao Coordenador o ajuste do tempo de mitigação para os parâmetros de comunicação.
43. wasRead	Informa ao Controlador que uma mensagem que foi enviada ao agente foi lida. Caso o agente não leia as mensagens encaminhadas a ele durante uma seção de trabalho, elas são novamente remetidas quando o usuário voltar a se logar no MultiJADE e trabalhar com o agente em questão.
44. getRestTime	Permite saber quanto falta para o término do tempo de mitigação de um conflito em torno de um parâmetro de comunicação.

Neste capítulo, abordamos as principais características de implementação. No próximo capítulo será mostrado um caso prático, focando principalmente no fluxo de mensagens no decorrer de um projeto no MultiJADE.

5 - ESTUDO DE CASO DO AMBIENTE MULTIJADE

O projeto de um veículo automotivo é bastante complexo, e requer que profissionais de diferentes áreas interajam em busca de um produto que satisfaça tanto as expectativas da empresa quanto às dos clientes.

Geralmente, os recursos disponíveis são limitados, e podem surgir impasses entre grupos que possuem diferentes visões e objetivos no projeto. Neste caso, o trabalho desenvolvido por estes grupos é concorrente. Sendo assim, é necessário que haja um processo de negociação de modo que o objetivo comum seja alcançado.

Para ilustração da aplicação do MultiJADE para resolução de problemas concorrentes e distribuídos, vamos nos centrar em dois grupos de projetistas: o primeiro responsável pela escolha dos pneus, e o segundo, pelo projeto da suspensão de um veículo.

Cada grupo de projetistas será representado por um agente específico e o coordenador será representado pela figura de um gerente encarregado do projeto. Vale destacar que nosso principal objetivo é evidenciar o uso e o fluxo de mensagens no MultiJADE.

5.1- Iniciando o MultiJADE

Para iniciar o MultiJADE, o usuário deve executar via linha de comando (java MultiJADE) a aplicação no seu diretório de instalação. Uma vez feito isto, é carregada a tela de login como mostrado na FIGURA 15.

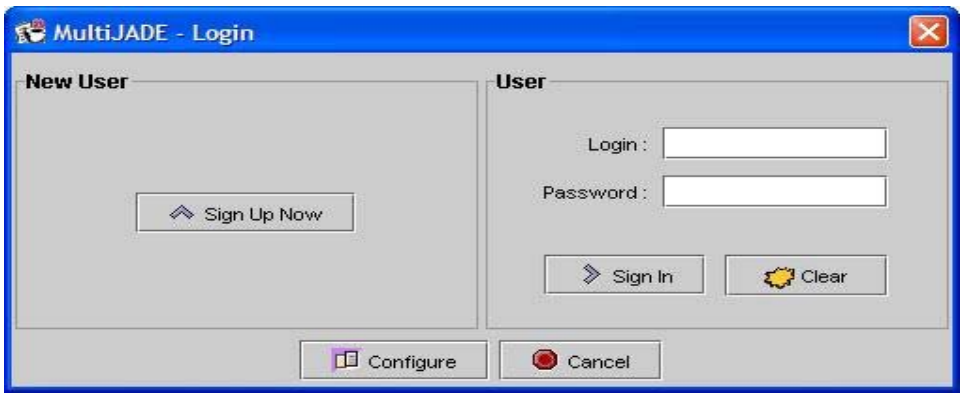

FIGURA 15 - Tela de login do MultiJADE.

Se o usuário ainda não está cadastrado, é necessário clicar em “Sign Up Now”, e preencher os campos requeridos. Assim feito, será chamada a primitiva addNewUser que tentará criar um novo usuário junto ao Servidor de Usuários. Caso o usuário já tenha se cadastrado previamente, basta preencher o login e a senha na tela de login, e clicar em “Sign In”. Então, será chamada a primitiva confirmUser que tentará fazer a autenticação do usuário. Bem sucedida qualquer uma das situações, o usuário terá acesso ao ambiente MultiJADE. Será considerado que todos os servidores foram iniciados como descrito no capítulo anterior.

5.2 - Criação de um modelo no MultiJADE

Através do ambiente MultiJADE, será criado um modelo com três agentes, que correspondem ao coordenador e aos grupos de especialistas responsáveis pela suspensão e pneus. Para isto, devemos acessar o menu File->Create Model. A FIGURA 16 exibe a tela de criação de modelo.

FIGURA 16 - Tela de criação de um modelo no MultiJADE.

Após a definição dos agentes e a identificação de qual deles é o coordenador, é preciso que o construtor do modelo defina os usuários dos agentes. Isto é feito através do botão “Agent Registry” (FIGURA 15). Cadastrados os usuários, ao clicar no botão “Ok”, é chamada a primitiva createModel, que cria um modelo na Biblioteca de Modelos.

A aquisição em um modelo no MultiJADE é subdividida em dois passos: a criação de parâmetros comuns e a criação de parâmetros específicos. Os parâmetros comuns são aqueles que estarão presentes no modelo de todos agentes. Já os parâmetros específicos fazem parte do domínio de agentes específicos, ou seja, não estão necessariamente presentes nos modelos de todos os agentes. Cabe ao criador do modelo a definição dos parâmetros comuns aos agentes.

Na FIGURA 17 são apresentados os parâmetros comuns aos dois agentes específicos do nosso modelo. Como podemos observar, temos como parâmetro primitivo a aceleração da gravidade e como parâmetros derivados a massa e o peso do veículo. Vale ressaltar que a escolha da Massa como parâmetro derivado só foi feita pelo fato de que este será um parâmetro de comunicação, e devido a limitações de implementação do MultiJADE, somente parâmetros derivados podem ser definidos como de comunicação.

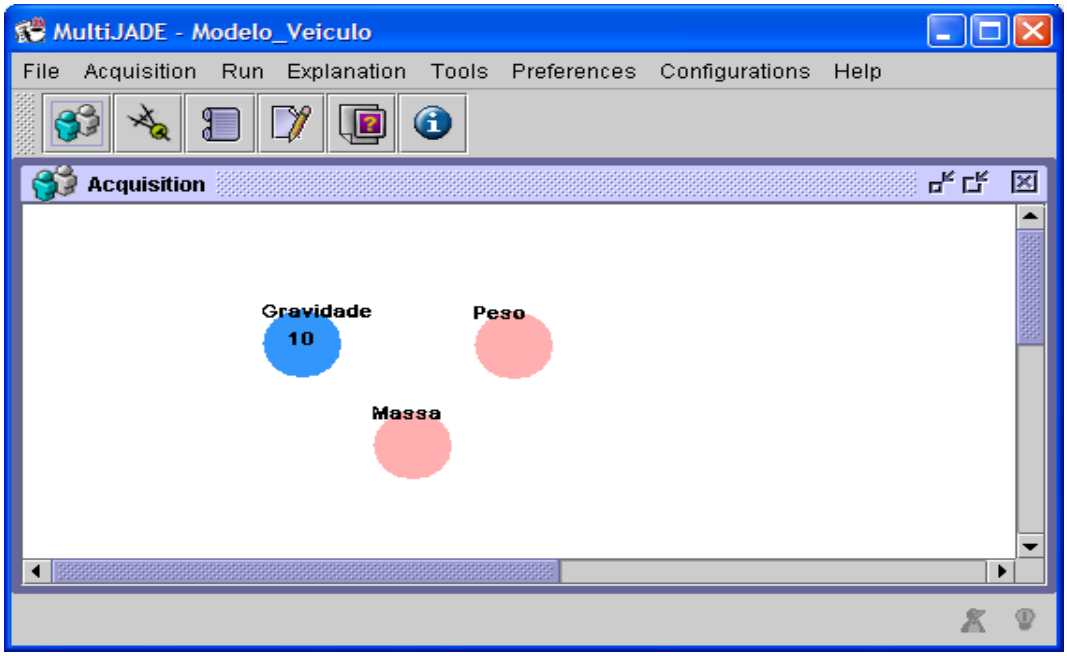

FIGURA 17 - Definição dos parâmetros comuns aos agentes

Após todos os parâmetros comuns serem adquiridos é necessário que o construtor do modelo defina quais são os parâmetros de comunicação. No nosso caso, o parâmetro de comunicação será a Massa, e sua configuração é feita através do menu Aquisition->Associate Parameter como exibido na FIGURA 18.

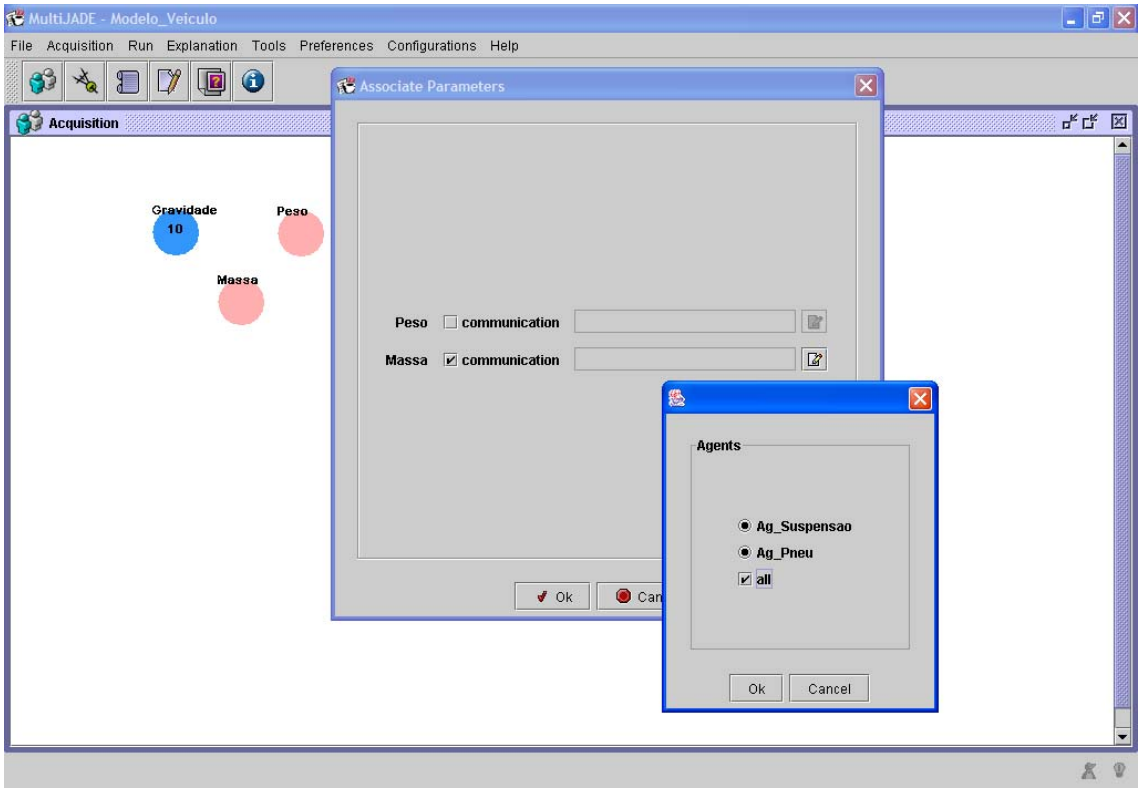

FIGURA 18 - Definição dos Parâmetros de Comunicação

Uma vez determinado o parâmetro de comunicação, é necessário definir qual será a relação deste parâmetro entre os agentes do projeto. Escolheremos a modo de exemplo, uma relação de igualdade. A escolha da relação para um parâmetro de comunicação nos agentes é feita através do menu Aquisition->Relation, como apresentado na FIGURA 19.

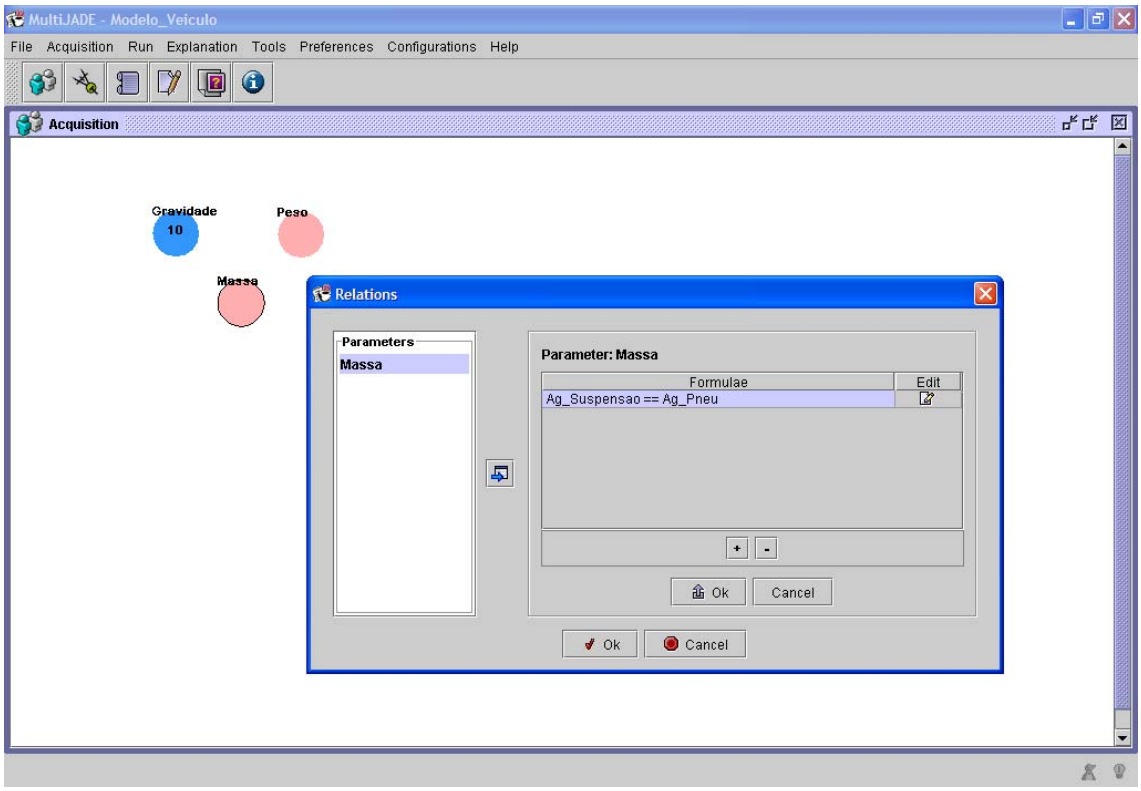

FIGURA 19 - Definição da relação do parâmetro de comunicação

Terminado este processo, o modelo deve ser salvo (primitiva uploadCreatingModel), e disponibilizado aos grupos que vão tratar das áreas específicas (primitiva parcialConcludeModel). Cabe destacar que todas as primitivas são chamadas de forma transparente ao usuário do MultiJADE.

A partir deste momento, os agentes específicos podem editar seus respectivos agentes. Para isto, é invocada a primitiva downloadCreatingAgent, para criar uma cópia local do agente.

No agente responsável pela suspensão, consideraremos que o objetivo é determinar a constante elástica da mola da suspensão. Para isto, será considerado que o peso que a mola suportará deve ser igual ao peso do carro. Assim, obtém-se a rede paramétrica exibida na FIGURA 20.

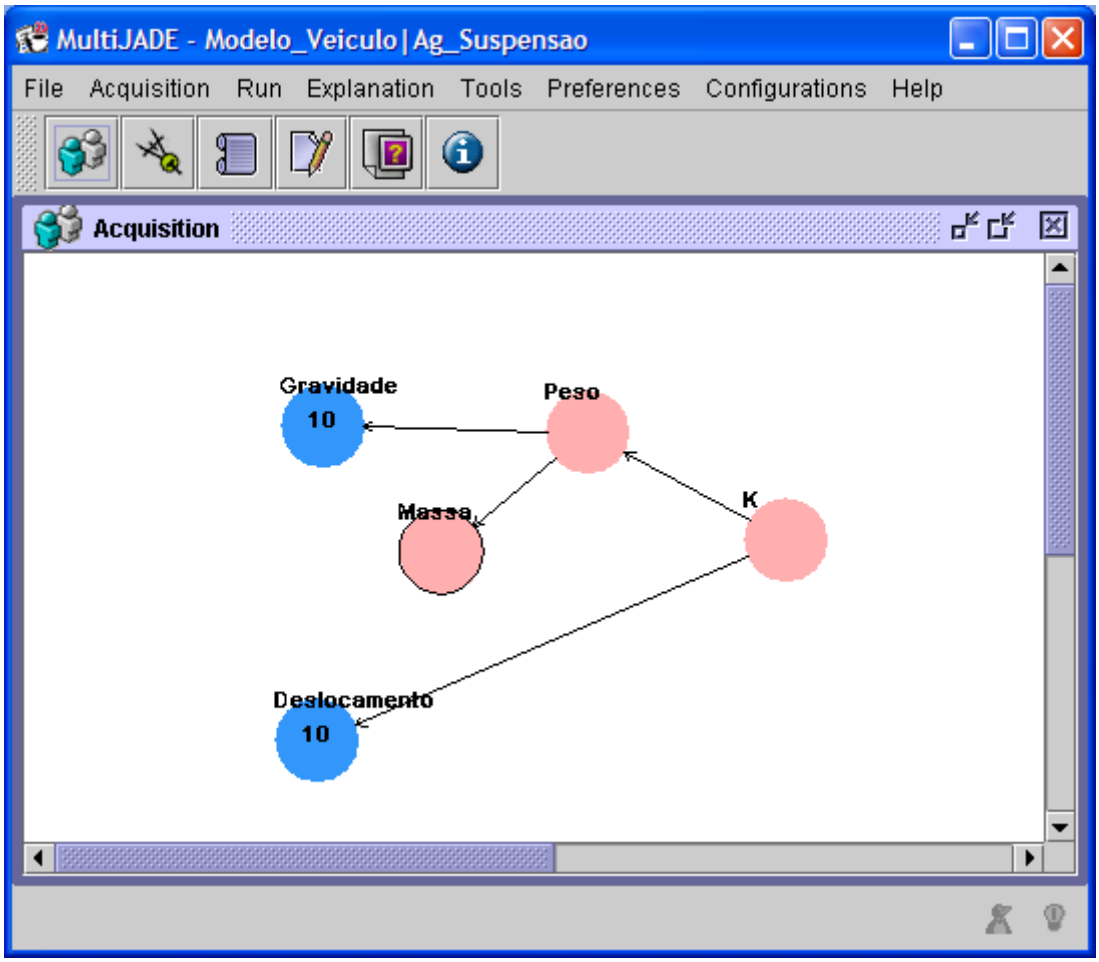

FIGURA 20 - Rede paramétrica do agente responsável pela suspensão.

O parâmetro Deslocamento representa quanto desejamos que a mola da suspensão seja comprimida. Neste exemplo adotamos o Deslocamento como sendo 10 cm. Já o parâmetro K representa a constante de elasticidade da mola que é obtida pela fórmula K = Peso/Deslocamento, onde Peso = Massa * Gravidade.

Uma vez criada a rede do agente responsável pela suspensão, é chamada a primitiva uploadCreatingAgent para salvá-lo, e a primitiva parcialConcludeAgent para concluir a edição do agente.

Os mesmos passos são seguidos para a aquisição da rede paramétrica do agente responsável pelo projeto dos pneus do carro. Neste caso, a rede paramétrica é exibida na FIGURA 21.

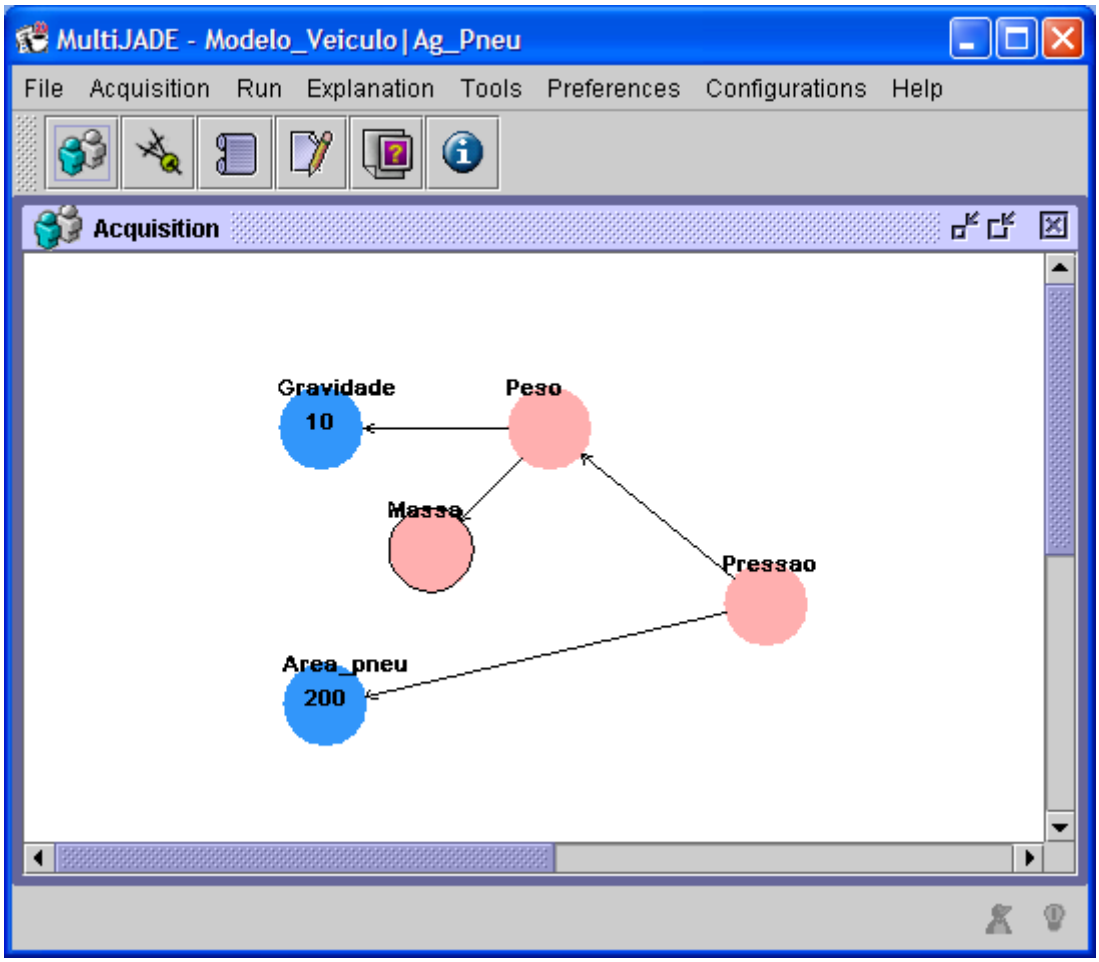

FIGURA 21 - Rede paramétrica do agente responsável pelos pneus

Neste agente, consideramos que o que queremos determinar é a pressão exercida sobre os pneus do carro. Para isto, temos que a pressão será dada por Pressão = (Peso)/Area_Pneu, onde Peso = Massa * Gravidade. Da mesma forma como foi levantado anteriormente, este agente deve ser salvo e concluído.

Por sua vez, o agente coordenador pode ser alterado como os outros agentes. Para isto os procedimentos descritos acima devem ser seguidos e o agente coordenador também deve ser concluído.

Ao término de todo este processo, o modelo é transferido do diretório de modelos em criação no Servidor de Biblioteca, para o diretório de modelos criados, e está pronto para ser utilizado na criação de um documento ativo.

5.3 - Criação de um documento ativo no MultiJADE

Como vimos, a criação de um documento ativo é baseada em um modelo. Deste modo, o projetista deve selecionar o modelo que será instanciado. A FIGURA 22 ilustra a interface de criação de um documento ativo, a qual pode ser acessada pelo menu File->Create Document do ambiente MultiJADE.

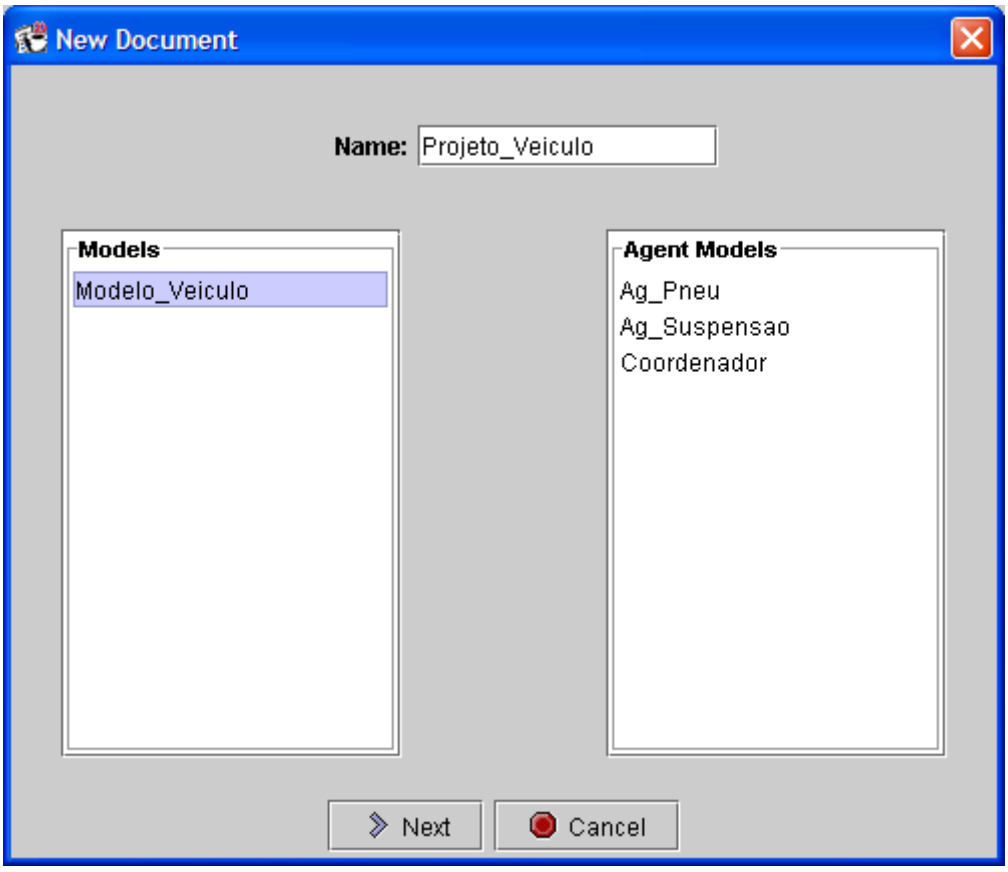

FIGURA 22 - Interface de criação de um Documento Ativo

Da mesma forma que na criação de um modelo, os usuários dos agentes do documento ativo também podem ser cadastrados.

Na criação de um projeto, é feita uma cópia de cada agente do modelo existente na biblioteca (primitiva downloadCreatedAgent), e logo depois, estas cópias são enviadas ao Controlador (primitiva firstUpload). Neste instante são registrados no Controlador os parâmetros de comunicação existentes em cada agente, assim como as relações que os regem.

Uma vez concluída a criação de um projeto no Controlador, os documentos ativos dos agentes estão prontos para serem utilizados por seus usuários no processo de *design*.

5.4 - Conflito no MultiJADE

Para evitar que num dado instante, várias pessoas possam manipular um documento ativo de um agente, sempre que um usuário tem a intenção de trabalhar em um projeto, o ambiente MultiJADE chama a primitiva setOnline. Se não houver ninguém utilizando aquele documento, ele é liberado e pode ser utilizado pelo usuário. Caso contrário, o usuário é notificado que outro usuário está trabalhando, e que naquele instante, não poderá acessar o agente.

Nosso objetivo será apresentar o fluxo de mensagens durante a mitigação de um conflito. Vamos considerar que o Ag_Pneu esteja trabalhando no dimensionamento dos pneus do nosso veículo. Em certo momento, o usuário atribui o valor do parâmetro Massa como sendo 1000Kg, como é exibido na FIGURA 23.

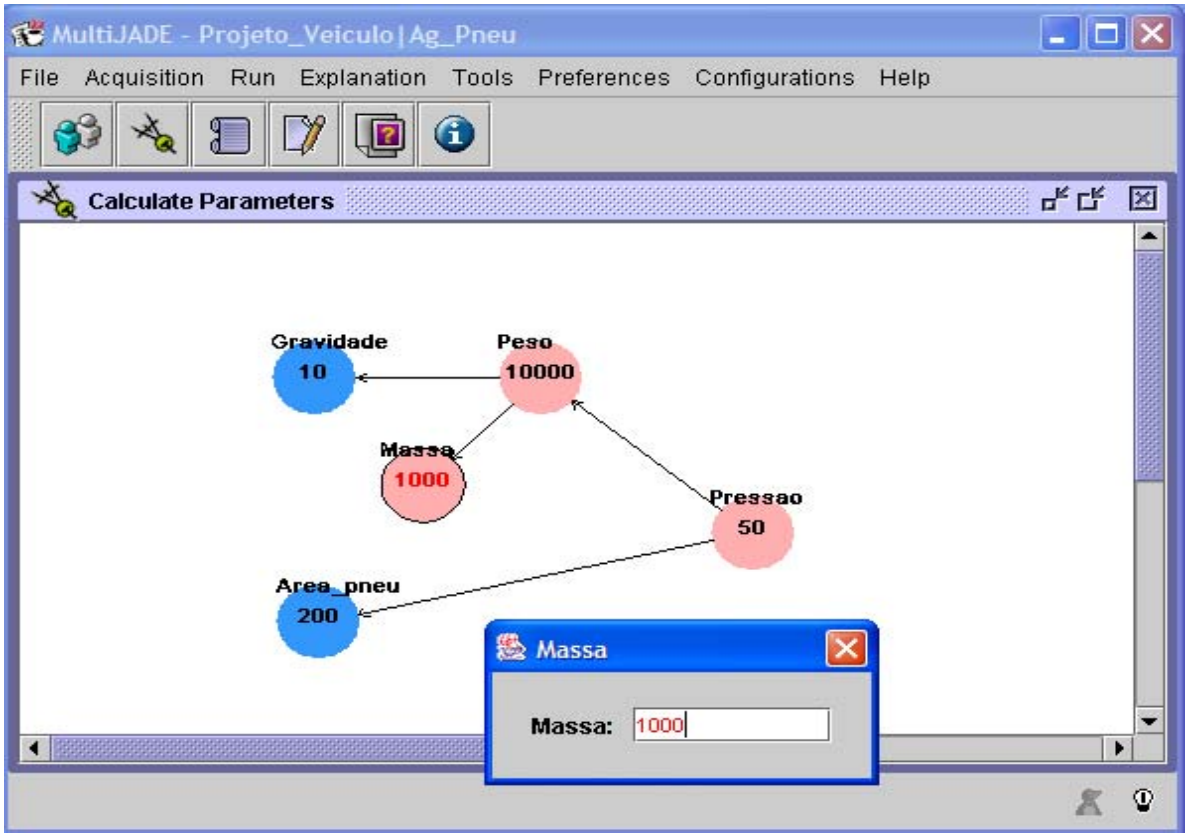

FIGURA 23 - Interface de alteração do valor de parâmetro

Neste momento, o ambiente chama, transparentemente ao usuário, a primitiva updateParameter que atualiza o valor deste parâmetro na base de dados do Controlador. Então, o Controlador sabendo que a relação em torno do parâmetro é de igualdade, tenta encaminhar uma mensagem ao agente Ag_Suspensao. Se este agente não estiver online naquele instante, a mensagem de notificação de mudança do parâmetro Massa no Ag_Pneu é armazenada para ser enviada quando aquele agente ficar online.

Uma vez online, o agente Ag_Suspensao é avisado que chegou uma nova mensagem através de um sinalizador que se encontra na parte inferior direita da janela do ambiente MultiJADE. Este sinalizador é apresentado na FIGURA 24.

FIGURA 24 - Sinalizador de novas mensagens

Para ler a mensagem, basta que o agente clique sobre a lâmpada e uma tela com as mensagens ainda não lidas endereçadas ao agente em questão irá aparecer.

Neste exemplo, a mensagem que chega ao Ag_Suspensao, é uma mensagem avisando que o Ag_Pneu modificou o parâmetro Massa. Além desta notificação, é questionado se o agente aceita ou não a sugestão feita. A mensagem de notificação é exibida na FIGURA 25.

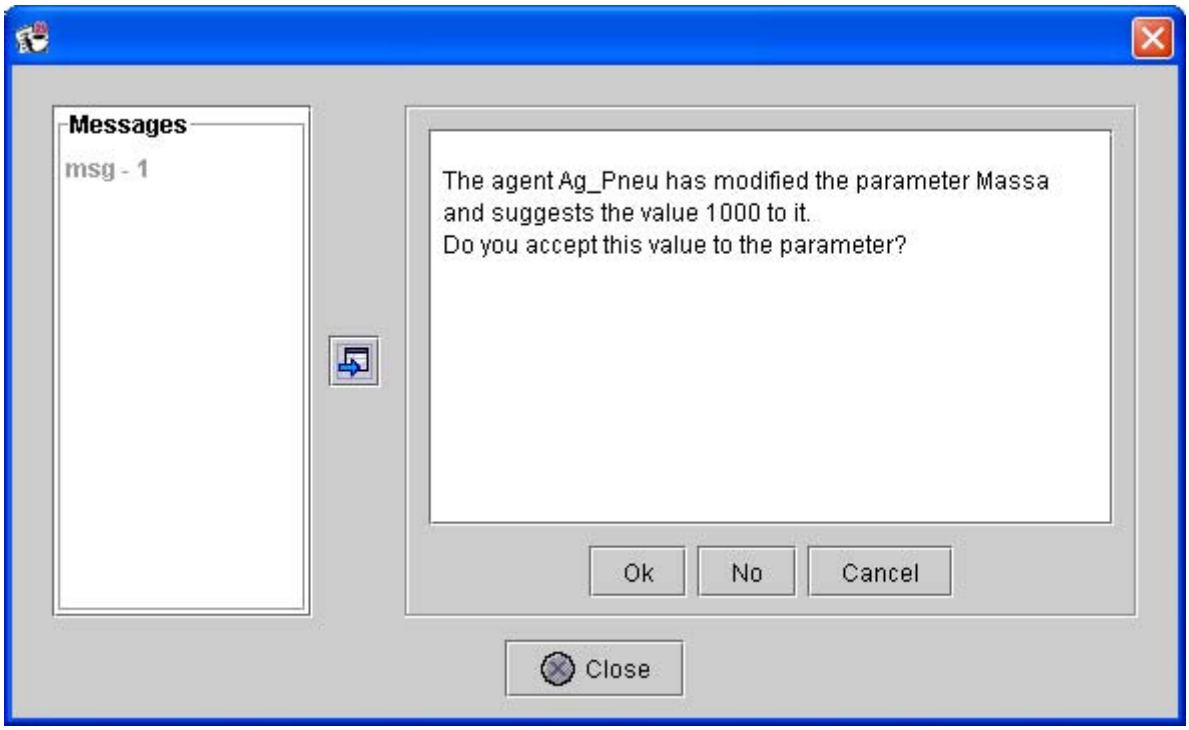

FIGURA 25 - Mensagem de sugestão de valor para parâmetro

Caso aceite a sugestão para o parâmetro Massa, o Controlador é notificado através da primitiva acceptChange, e tem-se mantida a situação de equilíbrio (sem conflito). Caso contrário, a primitiva refuseChange é invocada e o Controlador é comunicado que o Ag_Suspensao não aceitou tal modificação. Neste momento, o Controlador identifica a existência de um impasse e comunica a todos os envolvidos que existe um conflito em torno do parâmetro. A mensagem de notificação chega aos agentes no momento apropriado e indica quanto tempo eles terão para mitigar o conflito como exibido na FIGURA 26.

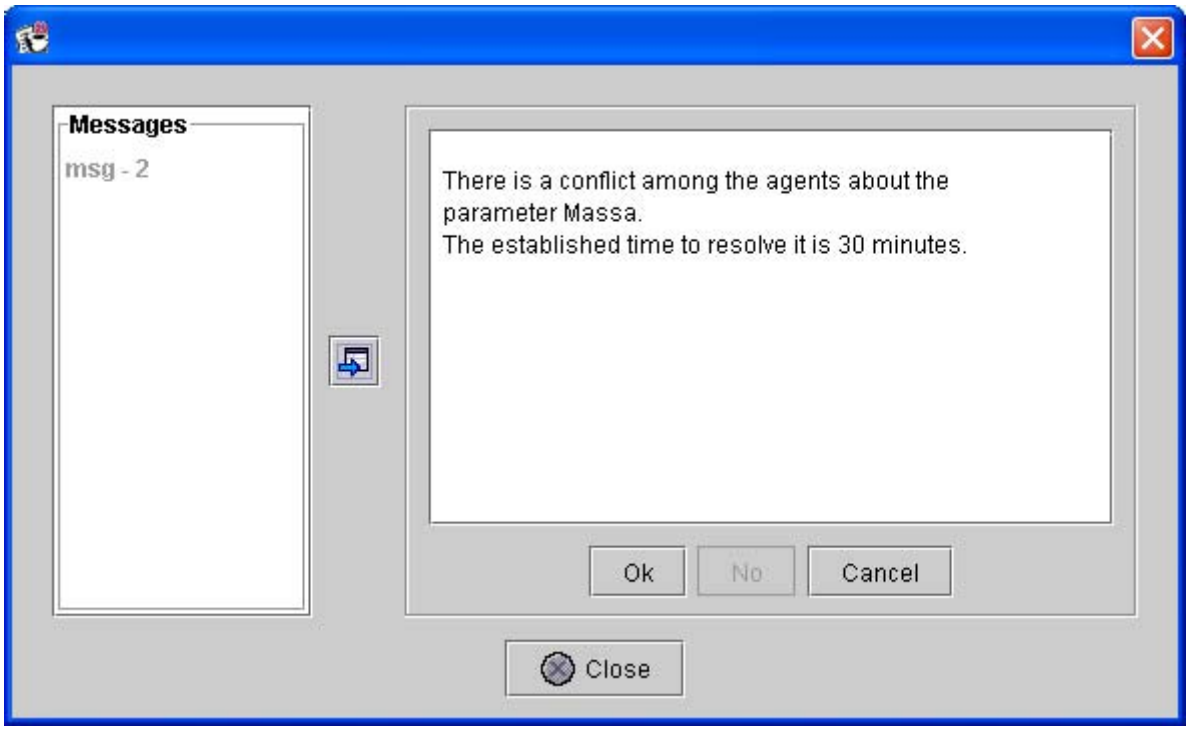

FIGURA 26 - Mensagem de notificação de conflito

Para auxiliar o processo de mitigação, os agentes podem utilizar recursos como seções de Chat, correio eletrônico e consulta de rationale. Durante a fase de negociação entre os agentes especialistas, cada valor sugerido para os parâmetros que gerou conflito é informado a todos os agentes envolvidos.

O processo de resolução de um conflito transcorre da seguinte maneira: ao chegarem a um consenso, um dos agentes modificará seu parâmetro na sua rede e os outros agentes serão notificados como apresentado anteriormente. Neste caso, os agentes aceitam a sugestão apresentada. Então, o Controlador envia uma mensagem com o objetivo de comunicar o término do conflito.

Em outra situação, podemos chegar a um estágio no qual não foi possível chegar a um acordo dentro do tempo de mitigação estabelecido para aquele parâmetro. Então, o Controlador identifica o término do tempo de mitigação e notifica ao Coordenador do projeto que existe um impasse em torno do parâmetro. Esta notificação pode ser vista na FIGURA 27.

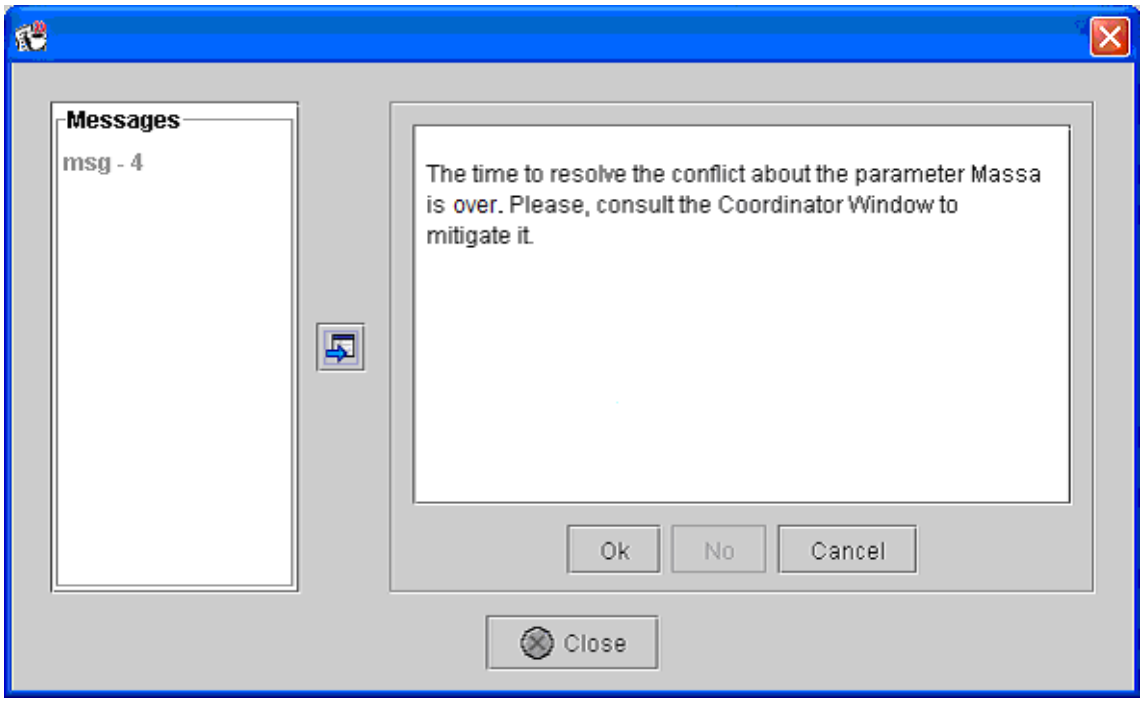

FIGURA 27 - Mensagem de notificação de persistência de conflito

Neste momento, o Coordenador do projeto deve analisar os motivos que levaram os agentes a tomarem suas decisões e moderar qual valor de parâmetro cada um ou ambos deve usar. Para isto, o Coordenador do projeto tem habilitada uma interface no seu ambiente de desenvolvimento onde ele pode configurar o valor do parâmetro para os agentes. Esta interface pode ser acessada pelo ícone que aparece no canto inferior direito ao lado do ícone de notificação de novas mensagens, como mostrado na FIGURA 28.

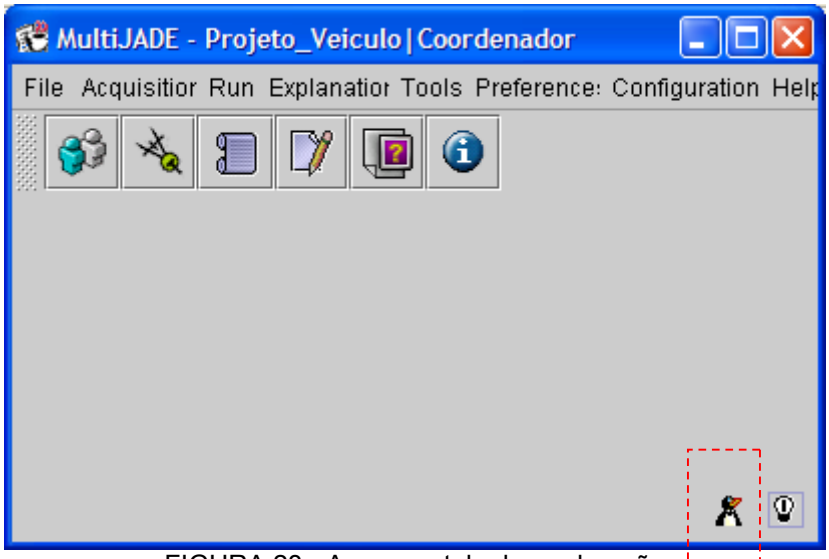

FIGURA 28 - Acesso a tela de moderação

Na interface de moderação de conflitos, o Coordenador do projeto pode além de definir valores para um determinado parâmetro nos agentes, configurar o tempo de mitigação. Esta interface é exibida na FIGURA 29.

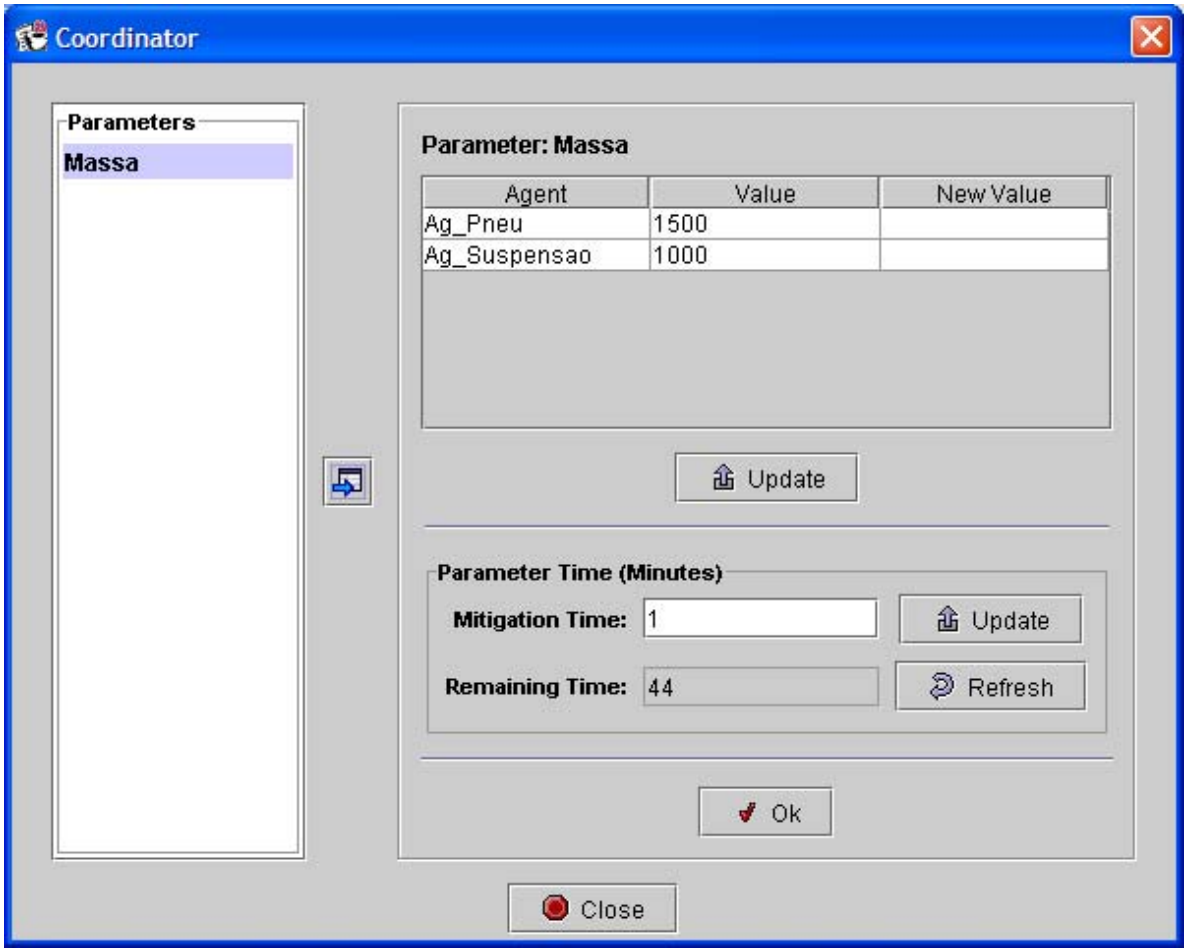

FIGURA 29 - Interface de moderação

É importante destacar que não cabe contestação a decisão tomada pelo Coordenador do projeto. Assim, após sua decisão, é estabelecida uma situação de equilíbrio, ou seja, fim do impasse.

Conforme foi dito, o MultiJADE não é um sistema automático para resolução de conflitos. Algumas de suas funções são atuar como identificador dos tipos de conflitos ocorridos e avisar aos agentes a respeito dos mesmos, facilitando a resolução dos problemas.

Este capítulo objetivou apresentar uma possível aplicação do MultiJADE e o fluxo de mensagens no decorrer da criação e desenvolvimento de um projeto. No próximo capítulo serão apresentadas as conclusões deste trabalho.

6 - CONCLUSÃO

O MultiJADE surgiu como uma proposta que visa utilizar a tecnologia da informação para facilitar a resolução de problemas concorrentes e distribuídos [Gama, 2004]. No entanto, tem a particularidade de não se propor a ser um solucionador automático de conflitos e sim dar suporte a resolução destes. Isso é possível uma vez que o MultiJADE possui um mecanismo de identificação de conflitos que comunica imediatamente os envolvidos e possibilita que eles tenham maior poder de negociação, visto que as razões das decisões dos agentes podem ser consultadas a qualquer momento, sem a necessidade de reuniões presenciais.

Para isto, o MultiJADE utiliza a tecnologia de sistemas multiagentes e de documentos ativos de design, e se baseia na proposta do MultiADD (uma abordagem para construção de sistemas multiagentes para *design* concorrente). Porém, o MultiADD consegue tratar apenas conflitos relativos a equivalência de valores. O MultiJADE pode tratar conflitos relativos a equivalência de valores (relações de igualdade) e conflitos que envolvam quebra das regras associadas aos parâmetros (relações gerais).

O sistema MultiJADE objetiva reduzir consideravelmente o esforço e o custo requeridos para a construção de sistemas MultiADD, de forma a minimizar as barreiras de utilização destes sistemas. Para isto, o MultiJADE provê um ambiente capaz de permitir que os usuários construam modelos multiagentes de forma rápida e intuitiva. Estes modelos podem ter domínios diversos e funcionam como base para construção de diversos documentos ativos. O fato de ser uma ferramenta ADD multiagente independente de domínio é um grande diferencial do MultiJADE com relação à proposta do MultiADD.

A metodologia que foi empregada na execução deste projeto de graduação envolveu uma ampla análise da literatura em processos concorrentes de tomada de decisão, a caracterização precisa do problema dos processos de tomada de decisão concorrente e distribuída, a implementação de toda a infraestrutura de gerenciamento e comunicação de dados de um ambiente computacional multiagente e distribuído baseado na proposta de uma

arquitetura centralizada e o teste deste sistema através da sua utilização por usuários da aplicação desenvolvida. Vale ressaltar que os módulos deste projeto foram desenvolvidos na linguagem de programação Java, assim como já vinha sendo feito o ambiente JADE.

As respostas encontradas têm sido bastante satisfatórias, visto que, a parte de comunicação entre os diversos agentes tem atendido a demanda de testes. Toda mudança em uma variável compartilhada é prontamente enviada ao agente afetado se este estiver online. Se ele estiver off-line, esta mensagem é guardada até que o mesmo volte a ativa.

A proposta de sala virtual, na qual determinados agentes podem resolver seus diferentes tipos de questões e conflitos, foi minuciosamente analisada e implementada levando-se em conta todas as funcionalidades desejadas. Essa sala virtual funciona como um *Chat* no qual os agentes podem entrar em uma sala e conversar em busca de um consenso. Caso não o alcancem, um administrador geral (gerente ou coordenador responsável) terá o poder de arbitrar, decidindo qual a solução mais viável para o impasse.

As principais dificuldades encontradas estavam relacionadas à integração do sistema de comunicação com o ambiente JADE para elaboração do MultiJADE. Isto se deve ao fato de que esta fusão devia ser a menos custosa possível, já que não devia alterar características do ambiente previamente construído. Esse processo foi bastante difícil e exigiu uma profunda análise do que realmente era necessário mudar. Outra dificuldade ficou por conta do grande esforço computacional envolvido em toda a elaboração do protocolo de comunicação necessário para o funcionamento do MultiJADE.

É importante frisar que a customização do cliente de e-mail [Ehnbom, 2000] e do sistema de *Chat* [McLaughlin] para que fossem alcançadas as características do sistema MultiJADE demandaram grande esforço, pois foi necessário um minucioso estudo do conjunto de classes destes softwares.

Para avaliar e validar os resultados do projeto de pesquisa foram realizados testes no sistema implementado. O ambiente computacional vem demonstrando a viabilidade de se criar um ambiente genérico para a construção de documentos ativos para um novo tipo de aplicação. A

implementação de um sistema de documentação ativa específico é um estudo de caso que demonstra a viabilidade de se utilizar este ambiente para construção de sistemas de documentação ativa para apoio aos processos de tomada de decisão concorrentes e distribuídos.

Vale lembrar que o MultiJADE herdou todas as características do JADE. A sua versão atual tem implementado somente a arquitetura centralizada que suporta uma considerável gama de problemas. Porém, como próximos passos deste projeto, podem ser implementadas outras arquiteturas como a hierárquica e a com total interação [Gama, 2004].

Na arquitetura hierárquica (vide FIGURA 30), temos uma sociedade composta por vários grupos de agentes. Esses grupos podem ou não ter um objetivo similar ao outro grupo. No entanto, todos os grupos cooperam entre si de forma a alcançar o objetivo da sociedade. Nesta arquitetura, cada grupo possui um coordenador responsável por auxiliar na resolução de eventuais conflitos e um agente do grupo não tem autonomia para negociar ou se relacionar com um agente especialista de um outro grupo. A negociação entre grupos é feita entre os agentes coordenadores dos grupos. Um subconjunto da arquitetura hierárquica equivale ao modelo de arquitetura centralizada.

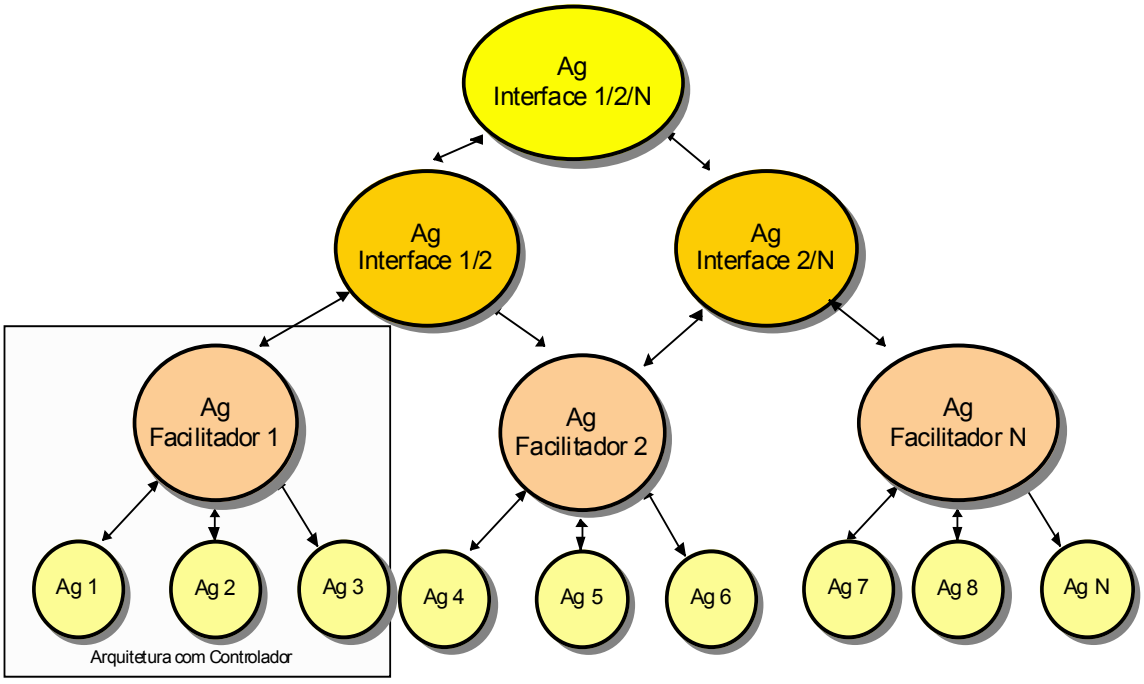

FIGURA 30 - Arquitetura hierárquica do MultiJADE

Na arquitetura com total interação (vide FIGURA 31) não existe a figura do coordenador. Neste caso, os agentes têm total autonomia para agir e resolver os conflitos. Isto demanda maior capacidade de negociação entre eles e cuidado para que o conflito não vire um impasse sem fim.

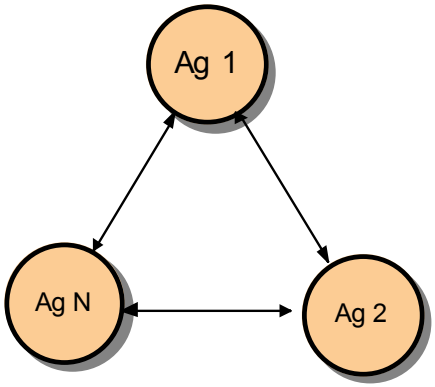

FIGURA 31 - Arquitetura MultiJADE com interação total

Uma funcionalidade do MultiJADE que poderá ser aprimorada é a resolução de conflitos, que atualmente é feita pelo agente humano. Trabalhos futuros podem investigar diferentes maneiras para resolução de conflitos ou sugestão automática de solução.

7 - REFERÊNCIAS

Braun, C. S., Varejão, F. M. Geração de Explicação no JADE. Universidade Federal do Espírito Santo, 2003.

Christopher, T. W., Thiruvathukal, G. K. High-Performance Java Platform Computing - Multithreaded and Networked Programming. Published by Sun Microsystems/Prentice Hall, 2001.

Ehnbom, F. Yet Another Mail Manager - YAMM http://www.gjt.org/~fredde/yamm, 2000.

Finger, S., Konda, S., Subrahmanian, E., Concurrent Design happens at the Interfaces, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 1995.

Gama, F. A. MultiJADE: um ambiente de auxílio a resolução distribuída de conflitos, baseado em documentos ativos. Mestrado em Informática, Universidade Federal do Espírito Santo, 2004.

Gama, F. A.; Varejão, F. M.; Guimarães, R. L.; Braun, C. S. MULTIJADE - a domain independent multiagent active design documents environment. First International Conference on Design Computing and Cognition (DCC'04), Boston - Massachusetts. Proceedings of the First International Conference on Design Computing and Cognition - DCC'04. John S. Gero: Kluwer Academic Publishers, p. 479-497, 2004.

Garcia, A. C. B. Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design, PhD thesis, Stanford University, United States, 1992.

Garcia, A. C. B., Vivacqua, A. S. MultiADD: A Multiagent Active Design Document. Proceedings of AAAI-97, Providence, Rhode Island, USA, pp.1066-1071, 1997.

Lee, J., Design Rationale Systems: Understanting the Issues, AI in Design, 1998.

McLaughlin, A. Babylon Java Chat. http://visopsys.org/andy/babylon/index.html.

Smith, Lan F. C. Conflict Management in Design, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 1995.

Varejão, F. M., Braun, C. S. Melhorando a Interação Humano-Computador no JADE. Publicado nos anais do Encontro Nacional de Inteligência Artificial (ENIA 2003), Campinas, 2003.

Varejão, F. M., Garcia, A. C. B. e Souza, C. S., Aquisição de Design Rationale através de Anotações Semi-Formais em Documentos Ativos de Design, 1996.

Varejão, F. M., Pessoa, R. M. Jade: A Computational Environment for Constructing, Using and Reusing Active Design Documents. Publicado nos anais do 5th IFIP International Workshop on Knowledge Intensive CAD, Malta, 2002.

Varejão, F. M., Pessoa, R. M. Um Ambiente Integrado para Construção, Uso e Reuso de Documentos Ativos de Design. Publicado nos anais do Encontro Nacional de Inteligência Artificial (ENIA 2001), Fortaleza, 2001.