4.
Raciocínio sobre Metas Flexíveis em Tempo de Execução

Esse capítulo descreve o segundo grande desafio para se obter a transparência de software: capacitar agentes de software intencionais a raciocinar em termos de metas flexíveis em tempo de execução. Dessa forma, o agente pode analisar seu próprio modelo e tomar decisões replicando o trabalho de um engenheiro de requisitos ao analisar um modelo intencional. Assim, garante-se que a lógica de execução do agente replica a intencionalidade modelada segundo o framework i. Apresentamos uma máquina de raciocínio qualitativa que aplica lógica nebulosa, uma teoria matemática para lidar com incertezas, para simular as regras de propagação utilizadas para analisar modelos i*.*

Existem muitas técnicas que apóiam os requisitos funcionais, provendo as funcionalidades desejadas. Entretanto, requisitos não-funcionais (RNFs) são difíceis de tratar em muitos projetos. Um dos primeiros *frameworks* a trabalhar com RNFs foi o NFR Framework (Chung et al. 2000). Esse *framework* – dentre outras contribuições – modela RNFs usando grafos específicos – Grafos de Interdependência entre Metas Flexíveis (*Softgoals Interdependence Graph* - SIG). O NFR Framework analisa RNFs aplicando regras de propagação a esse grafo, especialmente durante a Engenharia de Requisitos e com a participação dos interessados. Nesse contexto, os impactos das decisões são propagados pelo grafo usando uma análise qualitativa e visando determinar o quão bem uma escolha – feita em tempo de desenho – *satisfices* os RNFs analisados. De acordo com (Yu 1995), *satisfices* significa "satisfaz até um determinado grau", ou seja, satisfaz suficientemente. Dessa forma, o NFR Framework é um dos primeiros esforços da comunidade de Engenharia de Requisitos para lidar com RNFs – i.e. critérios de qualidade (e.g. precisão, desempenho e segurança) através do Processo de Desenvolvimento de Software, apresentando uma abordagem sistemática e pragmática – "contruindo qualidade dentro" (Chung et al. 2000) do software.

Outro *framework* que explora o uso de regras de propagação é o *framework* i* (Yu 1995; Yu 1997; ISTARWIKI 2011). Esse *framework* oferece uma abordagem orientada à meta para Engenharia de Requisitos, modelando as

estratégias de múltiplos atores. A principal ideia é entender e analisar contextos sociais visando melhorar o sucesso de um sistema. Novamente, essa tarefa é comumente realizada durante a Engenharia de Requisitos. Basicamente, os atores em modelos i* – *Strategic Dependency* (SD) e *Strategic Rationale* (SR) – dependem um do outro para que objetivos possam ser atingidos através de tarefas específicas. Nesse processo, recursos podem ser trocados entre os atores. Além disso, esse *framework* introduz a noção de meta flexível, que é usada como um critério de qualidade ou RNF. É também possível especificar as dependências entre atores, as quais são analisadas usando uma análise qualitativa centrada em regras de propagação. Nesse sentido, os atores – supervisionados por engenheiros de requisitos em tempo de desenho – exploram configurações alternativas de dependências para definir suas estratégias no contexto social ou no sistema multi-agente em análise.

Mais recentemente, o Tropos (Bertolini et al. 2006) – i.e. uma metodologia orientada a agente intencional para desenvolvimento de software centrado nos princípios da GORE (Van Lamsweerde 2001) – incorpora a noção de desenvolvimento dirigido por modelo evoluindo modelos i* (modelos SD e SR) da disciplina de *Early-Requirements* para a disciplina de Desenho Detalhado. Entretanto, Tropos também usa modelos i* para lidar com requisitos de software *early* e *late,* bem como desenho arquitetural e desenho detalhado. Depois desse processo, o software é finalmente implementado.

De acordo com (Bratman 1999), agentes inteligentes enriquecidos pelo conceito de intencionalidade são mais apropriados para lidar com o raciocínio prático humano e a formação de metas. Além disso, esse autor argumenta que é possível melhorar a capacidade cognitiva dos agentes usando uma máquina de raciocínio intencional focada em metas e estratégias específicas para atingi-las. Existem algumas abordagens baseadas no conceito de intencionalidade, tais como o modelo BDI (Bratman 1999; Bigus e Bigus 2001; Braubach et al. 2003), replicando atitudes mentais humanas que geram ações.

Entretanto, essa máquina de raciocínio intencional, que é a base do modelo BDI, considera apenas metas bem definidas (rígidas), ou *booleanas* – a meta é atingida ou não atingida, satisfeita ou não satisfeita. Todas as plataformas de SMAs que implementam o modelo BDI, como a plataforma JADE (Bellifemine et al 2007) com o *add-on* JADEX (Braubach et al. 2003; Braubach et al. 2004;

Braubach et al. 2005) ou o JACK (Busetta et al. 2011), seguem essa mesma filosofia para suas máquinas de raciocínio. Assim, nenhuma plataforma de SMAs intencionais, disponível atualmente, considera metas flexíveis, uma das principais abstrações do modelo i*.

A incapacidade dos agentes intencionais em lidar com metas flexíveis tornou-se um dos principais desafios em nosso objetivo de anexar os requisitos ao código. As metas flexíveis, os impactos positivos ou negativos das tarefas sobre elas, as interdependências entre metas flexíveis, as dependências de tarefas em metas flexíveis, entre outros, constavam no modelo de requisitos, mas simplesmente inexistiam no código do SMA intencional. Essas informações eram utilizadas pelos engenheiros de requisitos para tomar decisões em tempo de desenho, o que reduzia a variabilidade das estratégias dos agentes e limitava suas capacidades cognitivas em tempo de execução.

Ao invés de se trabalhar com RNFs e metas flexíveis em tempo de desenho, a proposta descrita nesse capítulo combina os conceitos previamente apresentados (ex. regras de propagação e SMAs intencionais) visando lidar com RNFs e metas flexíveis em tempo de execução. Observando, por exemplo, como a comunidade de Engenharia de Requisitos trata a análise de requisitos usando regras de propagação, desenvolvemos um simulador de propagação (Serrano et al. 2011) baseado em um algoritmo específico. Esse simulador tenta replicar as práticas dos engenheiros de requisitos quanto ao uso de regras de propagação na tomada de decisão em tempo de desenho. Nesse sentido, usamos a Teoria de Lógica Nebulosa (Zadeh 1965) para determinar o grau de satisfação de metas flexíveis enquanto são analisados os impactos e as dependências das metas flexíveis envolvidas, levando em consideração os relacionamentos estratégicos especificados entre os atores do sistema em desenvolvimento. Uma descrição detalhada do nosso simulador é apresentada na Seção 4.1 desse capítulo.

As heurísticas transformacionais apresentadas no capítulo anterior, o Capítulo 3, transformam as metas flexíveis, os impactos das tarefas, as interdependências entre metas flexíveis e outros em crenças dos agentes. Assim, essas informações estão disponíveis para os agentes. Com base no simulador de propagação proposto, desenvolvemos uma máquina de raciocínio qualitativa (Serrano et al. 2011) para SMAs intencionais capaz de analisar metas flexíveis em tempo de execução, selecionando uma estratégia adequada (i.e. um plano

adequado) que será desempenhado pelo agente intencional para atingir a meta desejada. Essa máquina de raciocínio combina tecnologias tradicionais e emergentes, tais como o NRC FuzzyJ Toolkit (Orchard 2006) e o *framework* JADEX (Braubach et al. 2004). Além disso, essa máquina tenta lidar com situações imprevisíveis, ou seja, que variam a qualquer momento, o que demanda uma análise em tempo de execução em detrimento de uma análise em tempo de desenho. A máquina de raciocínio proposta é detalhadamente discutida na Seção 4.2 desse capítulo. O código-fonte da máquina de raciocínio nebulosa para metas flexíveis está disponível integralmente no Apêndice B.

Esse capítulo está organizado da seguinte forma: a Seção 4.1 descreve o simulador de propagação; a Seção 4.2 apresenta a máquina de raciocínio proposta; na Seção 4.3, são discutidos os trabalhos relacionados bem como é feita uma comparação dos mesmos com a nossa abordagem; e finalmente, as considerações finais são apresentadas na Seção 4.4.

4.1. O Simulador de Propagação

Como apresentado anteriormente, o NFR Framework (Chung et al. 2000) propõe uma abordagem para analisar grafos de RNFs usando regras de propagação. Atualmente, essa abordagem é considerada um padrão na comunidade de Engenharia de Requisitos, sendo também utilizada no *framework* i* (Yu 1997). Dessa forma, os engenheiros de requisitos adquiriram a prática em aplicar regras de propagação para analisar modelos de requisitos.

Foi observado como os engenheiros de requisitos trabalham com regras de propagação em dois grupos de pesquisa de Engenharia de Requisitos: o grupo de Engenharia de Requisitos da PUC-Rio e o grupo de Engenharia de Software da *University of Toronto* (UofT). Com base nessas observações, tentamos replicar essas práticas em um simulador de propagação (Serrano et al. 2011).

O simulador de propagação proposto lida com RNFs e metas flexíveis. Como definido pelo NFR Framework, RNFs são critérios de qualidade esperados pelos interessados e que devem ser considerados no desenvolvimento do software. Critérios de qualidade são subjetivos, não facilmente mensuráveis através de métricas. Por exemplo, não faz sentido dizer que um determinado software é

totalmente seguro. Metas flexíveis são metas que não podem ser claramente atingidas. Por exemplo, não é possível dizer que a meta flexível "Desempenho[Software]" foi satisfeita. O que poderia ser dito é que o desempenho do software é bom o suficiente. Como ambos os conceitos – RNFs e metas flexíveis – envolvem uma incerteza intrínseca, parece apropriado representá-los usando lógica nebulosa (Zadeh 1965). Lógica nebulosa é uma teoria matemática que lida com incertezas. O simulador proposto aplica lógica nebulosa visando simular o grau de satisfação dos RNFs ou metas flexíveis.

Primeiramente proposta por Zadeh, a lógica nebulosa (Zadeh 1965) difere da lógica discreta, onde conjuntos binários possuem elementos *booleanos* (i.e. membro ou não membro). Ao contrário dessa lógica, a lógica nebulosa define uma função de pertinência para um conjunto. Por exemplo, é possível dizer que uma pessoa com 1,80 metros de altura é uma pessoa alta com 70% de certeza, caso a função de pertinência retorne 0.7 para o valor de entrada 1,80. Acabamos de explicar o primeiro conceito da lógica nebulosa: os conjuntos nebulosos. Outro conceito é a variável nebulosa, o qual representa o domínio – no caso do exemplo, alturas das pessoas (em metros) – e os conjuntos nebulosos conhecidos para esse domínio: pessoas baixas, pessoas médias e pessoas altas. Valores nebulosos são também representados por funções de pertinência e servem como valores para as variáveis nebulosas.

De acordo com (Chung et al. 2000) e (Yu 1995), RNFs e metas flexíveis têm grau de satisfação de 100% negativo (totalmente recusado, ou *denied*) a 100% positivo (totalmente satisfeito, ou *satisfied*). Existem algumas expressões linguísticas como *denied*, *undecided*, *partially_satisficed*, *satisficed* que podem ser aplicadas para descrever o grau de satisfação de um RNF ou meta flexível. Essas expressões linguísticas são claramente candidatas a conjuntos nebulosos. A Figura 4.1 mostra nossa representação de graus de satisfação de RNFs ou metas flexíveis com variáveis nebulosas (domínio mais conjuntos nebulosos). As áreas desses conjuntos são definidas usando bom senso e empiricamente ajustadas, como é usual quando lidamos com a teoria de lógica nebulosa.

Adicionalmente, foram definidas como variáveis nebulosas as contribuições a partir das operacionalizações ou tarefas associadas aos RNFs ou metas flexíveis. Os conjuntos nebulosos foram definidos com base nas expressões

linguísticas *break*, *hurt*, *help*, e *make*, definidas como elos de contribuição no *framework* i* (Yu 1997).

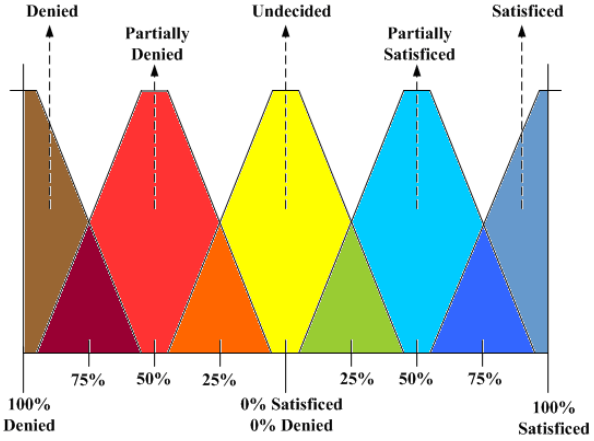


Figura 4.1 - Os graus de satisfação das metas flexíveis vistos como uma variável nebulosa.

A Figura 4.2 ilustra um modelo i* genérico com duas tarefas que podem atingir uma meta. A primeira tarefa contribui positivamente (elo de contribuição *Make*) para a meta flexível. A segunda tarefa contribui negativamente (elo de contribuição *Hurt*) para a meta flexível. Analisamos o modelo i* aplicando as regras de propagação da Tabela 4.1, extraídas do i* Wiki (ISTARWIKI 2011). Por exemplo, caso a primeira tarefa seja selecionada para atingir a meta, a execução dessa tarefa impactará positivamente (quadrado marcado em azul na Tabela 4.1) na meta flexível. Esse impacto positivo está representado pelo impacto *Satisficed* na lista de impactos. Como a primeira tarefa foi selecionada, a não execução da segunda tarefa impactará positivamente (quadrado marcado em verde na Tabela 4.1) na meta flexível. Esse impacto positivo está representado pelo impacto *Partially_Satisficed* na lista de impactos. Alguns pesquisadores não concordam com o impacto colateral gerado pela não execução da segunda tarefa, portanto, o simulador permite desabilitar os impactos colaterais.

Nosso objetivo é representar as regras de propagação apresentadas na Tabela 4.1 em um simulador baseado em lógica nebulosa. Para manipular (i.e. executar operações em) variáveis nebulosas, a lógica nebulosa aplica regras nebulosas, expressas como "SE variável É valor ENTÃO variável É valor". A propagação gerada pelo elo de contribuição *Make*, com base na Figura 4.2, pode ser representada como uma regra nebulosa: "SE contribuição É *make* ENTÃO meta flexível É *satisficed*". Dessa forma, supondo que a primeira tarefa da Figura

4.2 seja selecionada, o simulador avaliará as regras nebulosas e aplicará nossa regra caso a mesma seja condizente com a condição "contribuição É *make*".

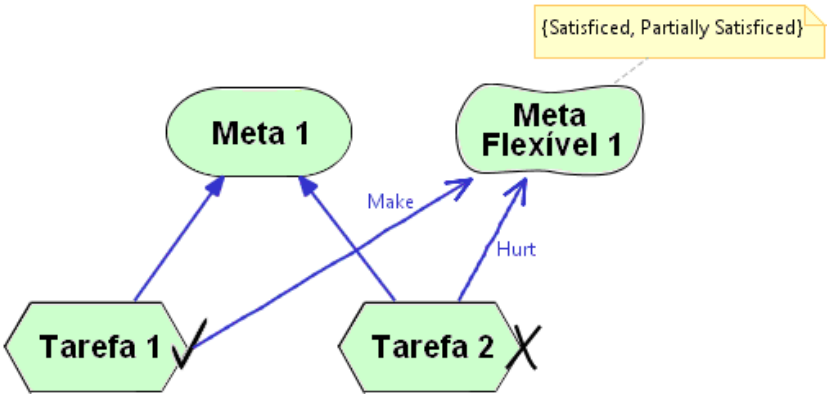


Figura 4.2 - Duas tarefas contribuindo para uma meta

Tabela 4.1 - Regras de propagação do *framework* i* (ISTARWIKI 2011).

Originating Label		**Contribution Link Type**			
Label	Name	Make	Break	Help	Hurt
✓	Satisficed	✓	✗	✓.	✗.
✓.	Partially Satisficed	✓.	✗.	✓.	✗.
↯	Conflict	↯	↯	↯	↯
?	Unknown	?	?	?	?
✗.	Partially Denied	✗.	✓.	✗.	✓.
✗	Denied	✗	✓.	✗.	✓.

Quando um NFR ou meta flexível é decomposto usando *AND* em duas ou mais metas flexíveis, seu grau de satisfação é o menor grau de satisfação das metas flexíveis "folhas". Metas flexíveis decompostas usando *OR* têm o maior grau de satisfação das metas flexível "folhas".

Com base na nossa representação de NFRs e contribuições como variáveis nebulosas e das regras de propagação como regras nebulosas, propomos o algoritmo apresentado a seguir. O algoritmo simula o trabalho do engenheiro de requisitos ao avaliar as estruturas meios-fim (*means-end structures*) do modelo. Nessas estruturas, temos uma meta, uma ou mais tarefas, e zero ou mais contribuições para as metas flexíveis.

```
// retorna o número da tarefa selecionada
inteiro simularPropagação (estrutura meios-fim) {
    // primeira parte – análise de contexto e inicialização
    Para cada tarefa recuperar as metas flexíveis impactadas;
```

Para cada meta flexível impactada criar uma variável nebulosa;
Para cada tarefa recuperar as contribuições;
Para cada contribuição criar uma variável nebulosa e atribuir à ela um valor nebuloso de acordo com o tipo do elo de contribuição;
Para cada contribuição criar regras nebulosas de acordo com o tipo de elo de contribuição;

// segunda parte – disparando as regras
Para cada tarefa {
Marcar a tarefa como selecionada (i.e. *satisficed*);
Marcar as outras tarefas como não selecionadas (i.e. *denied*);
Avaliar cada regra de propagação nebulosa e executar as regras compatíveis;
Para cada meta flexível impactada armazenar sua lista de impactos;
Salvar as listas de impactos das metas flexíveis impactadas como uma alternativa;
}

// terceira parte – avaliação
Para cada alternativa {
Para cada meta flexível {
Defuzificar a meta flexível;
Multiplicar o valor defuzificado pela prioridade da meta flexível e somar o resultado ao total da alternativa;
}
}

Retornar o número da tarefa correspondente à alternativa com o maior total;
}

Aplicando esse algoritmo no exemplo da Figura 4.2, temos que a primeira parte do algoritmo:

- Recupera a meta flexível “Meta Flexível 1” como a única meta flexível impactada;
- Cria uma variável nebulosa para representar o grau de satisfação da “Meta Flexível 1”;
- Recupera a contribuição *Make* da “Tarefa 1” e a contribuição *Hurt* da “Tarefa 2”;
- Cria uma variável nebulosa para a contribuição *Make* e atribui a ela o valor nebuloso “*make*”;
- Cria uma variável nebulosa para a contribuição *Hurt* e atribui a ela o valor nebuloso “*hurt*”;
- Cria as seguintes regras nebulosas para propagar os impactos através da contribuição *Make* da “Tarefa 1”:
  1. SE “Tarefa 1” É *denied* ENTÃO “Meta Flexível 1” É denied;
  2. SE “Tarefa 1” É *partially_denied* ENTÃO “Meta Flexível 1” É *partially_denied*;
  3. SE “Tarefa 1” É *partially_satisficed* ENTÃO “Meta Flexível 1” É *partially_satisficed*; e
  4. SE “Tarefa 1” É *satisficed* ENTÃO “Meta Flexível 1” É *satisficed*;
- Cria as seguintes regras nebulosas para propagar os impactos através da contribuição *Hurt* da “Tarefa 2”:
  5. SE “Tarefa 2” É *denied* ENTÃO “Meta Flexível 1” É *partially_satisficed*;
  6. SE “Tarefa 2” É *partially_denied* ENTÃO “Meta Flexível 1” É *partially_satisficed*;
  7. SE “Tarefa 2” É *partially_satisficed* ENTÃO “Meta Flexível 1” É *partially_denied*; e
  8. SE “Tarefa 2” É *satisficed* ENTÃO “Meta Flexível 1” É *partially_denied*.

A segunda parte do algoritmo:

- Analisa a primeira alternativa, marcando a “Tarefa 1” como selecionada (*satisficed*) e a “Tarefa 2” como não selecionada (*denied*);

- Avalia as oito regras nebulosas, acionando as regras nebulosas 4 e 5, impactando a "Meta Flexível 1" com os valores nebulosos {*satisficed*, *partially_satisficed*};
- Armazena a lista de impactos da "Meta Flexível 1" e salva a alternativa 1.
- Analisa a segunda alternativa, marcando a "Tarefa 1" comos não selecionada (*denied*) e a "Tarefa 2" como selecionada (*satisficed*);
- Avalia as oito regras nebulosas, acionando as regras nebulosas 1 e 8, impactando a "Meta Flexível 1" com os valores nebulosos {denied, partially_denied} (um resultado pior do que o da alternativa 1); e
- Armazena a lista de impactos da "Meta Flexível 1" e salva a alternativa 2.

A terceira parte do algoritmo:

- Defuzifica a "Meta Flexível 1" da alternativa 1;
- Multiplica o valor por 1.0 (prioridade *default*) e atribui o resultado ao total da alternativa 1;
- Defuzifica a "Meta Flexível 1" da alternativa 2;
- Multiplica o valor por 1.0 (prioridade *default*) e atribui o resultado ao total da alternativa 2;
- Compara ambas as alternativas; e
- Retorna o número da primeira alternativa, ou seja, seleciona a "Tarefa 1".

4.2. Máquina de Raciocínio Qualitativa

O Tropos (Bertolini et al. 2006, Castro et al. 2002) introduz um processo de desenvolvimento de software dirigido por requisitos, onde os requisitos modelados com o *framework* i* guiam a produção de outros artefatos. Inicialmente (Castro et al. 2002), o Tropos focou na produção de artefatos desenhados em UML e no software orientado a objetos. Mais recentemente (Bresciani et al. 2004), o Tropos aplicou o paradigma da orientação a agentes no processo dirigido a modelo, produzindo um SMA intencional. As noções metalinguísticas do modelo BDI (Bratman 1987) que aparecem no *framework* i* –

e.g. as metas e os planos – são também representadas no código dos agentes intencionais.

Similarmente a outros grupos de pesquisas, desenvolvemos heurísticas transformacionais (Serrano e Leite 2011b) que guiam a produção do desenho arquitetural, do desenho detalhado e da implementação dos agentes intencionais. Entretanto, notamos que todo o raciocínio relacionado aos RNFs, ou seja, as metas flexíveis e as contribuições recebidas, não são representadas no código. A análise dos RNFs, nas abordagens propostas por outros grupos de pesquisa, é feita em tempo de desenho, decidindo – nesse momento – quais tarefas precisam ser incluídas no software.

Apesar desse consenso entre as abordagens pesquisadas, o descarte de tarefas (meios de se atingir as metas ou fins) é contra-produtivo, uma vez que reduz a variabilidade de tarefas e, conseqüentemente, a quantidade de alternativas nas tomadas de decisões dos agentes intencionais. Uma tarefa que tem impactos negativos nas metas flexíveis pode ser a **única** forma encontrada para se atingir uma determinada meta em tempo de execução, no caso de todas as demais tarefas falharem. Dessa forma, a análise do RNF precisa ser realizada em tempo de execução pelo agente.

Nesta seção, é proposta uma máquina de raciocínio (Serrano et al. 2011) para agentes intencionais usando como base nosso simulador de propagação. A ideia é criar uma capacidade que poderia ser reutilizada por todos os agentes, permitindo a eles não apenas entender as noções do modelo BDI (como por exemplo: crenças, metas e planos), mas também metas flexíveis, contribuições e regras de propagação.

Nossos agentes intencionais são implementados usando o *framework* JADEX (Braubach et al. 2004), i.e. uma extensão baseada no modelo BDI para a plataforma JADE (Bellifemine et al. 2007). Conforme explicado no Capítulo 2, agentes JADEX são descritos usando um *Agent Definition File* (ADF) – um arquivo XML – e planos, os quais são implementados em classes Java que estendem a classe "Plan" do JADEX. O ADF define as metas, os planos, as capacidades, os eventos e outras noções do agente. Quando uma meta torna-se ativa, o JADEX seleciona um plano apropriado e executa-o. Caso dois ou mais planos sejam avaliados para atingir a meta, o JADEX ativa uma meta-meta que dispara um meta-plano. O JADEX não oferece o código do meta-plano, uma vez

que esse código precisa ser implementado pelo próprio desenvolvedor. Caso o agente intencional esteja habilitado com a capacidade da máquina de raciocínio qualitativa proposta (ou seja, adquira a capacidade), esse meta-plano já contém uma implementação da nossa máquina de raciocínio. A Figura 4.3 mostra a declaração da capacidade em um ADF.

```
<capability name="ReasoningEngine"
        file="istar.metamodel.PropagationSimulator"/>
```

Figura 4.3 - Habilitando o agente com a capacidade da máquina de raciocínio qualitativa.

Como os planos no JADEX são implementados em Java, nosso simulador de propagação foi implementado usando uma API de lógica nebulosa baseada em Java, nomeada NRC FuzzyJ Toolkit (Orchard 2006). Essa API oferece a implementação de conceitos de lógica nebulosa, tais como: variáveis, conjuntos, valores e regras nebulosas. A Figura 4.4 apresenta o atributo "fuzzyVariable" da classe "Softgoal" sendo instanciado como uma variável nebulosa que representa o grau de satisfação ilustrado na Figura 4.1.

```
FuzzyVariable fuzzyVariable1 =
    new FuzzyVariable(softgoalName, -100.0, 100.0, "percentage");

fuzzyVariable1.addTerm("denied",
    new TrapezoidFuzzySet(-100.0, -100.0, -95.0, -55.0));
fuzzyVariable1.addTerm("partially_denied",
    new TrapezoidFuzzySet(-95.0, -55.0, -45.0, -5.0));
fuzzyVariable1.addTerm("undecided",
    new TrapezoidFuzzySet(-45.0, -5.0, 5.0, 45.0));
fuzzyVariable1.addTerm("partially_satisficed",
    new TrapezoidFuzzySet(5.0, 45.0, 55.0, 95.0));
fuzzyVariable1.addTerm("satisficed",
    new TrapezoidFuzzySet(55.0, 95.0, 100.0, 100.0));

this.fuzzyVariable = fuzzyVariable1;
```

Figura 4.4 - A variável nebulosa (domínio e conjuntos nebulosos) de uma meta flexível

Os planos em JADEX não representam contribuições de tarefas para as metas flexíveis, uma vez que esse conceito não está presente no modelo BDI. No intuito de atingir esse propósito, implementamos as classes "Task" e "Contribution". A Figura 4.5 ilustra parcialmente a classe "Task". O nome da tarefa, seu vetor de contribuições para as metas flexíveis e o valor nebuloso são atributos da classe "Task". O valor nebuloso é usado pela segunda parte do algoritmo do simulador de propagações visando marcar a tarefa como "selecionada" ou "não selecionada". O código parcial referente à classe

"Contribution" é apresentado na Figura 4.6. O tipo de contribuição é um valor enumerado que pode ser "*break*", "*hurt*", "*help*", "*make*", "*some*+", e "*some*-".

```
public class Task {

    private String taskName;
    private Contribution[] softgoalContributions;
    private FuzzyValue fuzzyValue;

    {...}
}
```

Figura 4.5 - Código parcial para a classe "Task"

```
public class Contribution {
    private Task contributor;
    private ContributionType contributionType;
    private Softgoal contributed;

    {...}
}
```

Figura 4.6 - Código parcial para a classe "Contribution"

A primeira parte do algoritmo do simulador de propagação gera dinamicamente as regras de propagação nebulosas de acordo com o tipo de contribuição. A Figura 4.7 mostra o código de geração das regras nebulosas relacionadas ao tipo de contribuição "help". O método "addAntecedent" adiciona um valor nebuloso como condição da parte "SE" da regra nebulosa. O método "addConclusion" adiciona um valor nebuloso como o resultado – a parte "ENTÃO" da regra nebulosa. A variável local "contributor" contém o nome da tarefa. A variável local "contributed" contém o nome da meta flexível impactada. Note como essas quatro regras nebulosas estão relacionadas à coluna "help" da Tabela 4.1 (os *labels* "conflict" e "unknown" são ignorados).

```
case HELP:
    fuzzyRule1.addAntecedent(
            new FuzzyValue(contributor, "satisficed"));
    fuzzyRule1.addConclusion(
            new FuzzyValue(contributed, "partially_satisficed"));
    fuzzyRule2.addAntecedent(
            new FuzzyValue(contributor, "partially_satisficed"));
    fuzzyRule2.addConclusion(
            new FuzzyValue(contributed, "partially_satisficed"));
    fuzzyRule3.addAntecedent(
            new FuzzyValue(contributor, "partially_denied"));
    fuzzyRule3.addConclusion(
            new FuzzyValue(contributed, "partially_denied"));
    fuzzyRule4.addAntecedent(
            new FuzzyValue(contributor, "denied"));
    fuzzyRule4.addConclusion(
            new FuzzyValue(contributed, "partially_denied"));
    break;
```

Figura 4.7 - Código de geração de regras nebulosas para o tipo de contribuição "help".

Nossa máquina de raciocínio considera que o agente conhece o contexto, suas metas flexíveis, contribuições e tarefas. Isso significa que o agente precisa ter três conjuntos de crenças com os objetos metas flexíveis, contribuições e tarefas. Esses objetos precisam ser instanciados e inicializados no *setup* do agente. A declaração de crenças no ADF pode ser vista na Figura 4.8. A Figura 4.9 mostra a inicialização de uma tarefa com um elo de contribuição *break* para uma meta flexível.

```
<beliefs>
    <beliefset name="softgoals" class="Softgoal"/>
    <beliefset name="contributions" class="Contribution"/>
    <beliefset name="tasks" class="Task"/>
</beliefs>
```

Figura 4.8 - Metas flexíveis, contribuições e tarefas como crenças dos agentes

```
Task task1 = new Task("Task name");
Contribution[] softgoalsContributions1 =
    {new Contribution(task1,ContributionType.BREAK,softgoal1)};
task1.setSoftgoalContributions(softgoalsContributions1);
this.getBeliefbase().getBeliefSet("tasks").addFact(task1);
```

Figura 4.9 - Inicialização de uma tarefa e sua contribição no *setup* do agente

Dois tipos de metas flexíveis são avaliados pela máquina de raciocínio: metas flexíveis decompostas e metas flexíveis "folhas". Metas flexíveis decompostas são metas flexíveis decompostas por operadores *AND* ou *OR*, com duas ou mais metas flexíveis "filhas". Metas flexíveis "folhas" não são decompostas em outras metas flexíveis. Ambos os tipos de metas flexíveis estendem a classe abstrata "Softgoal", Figura 4.10. Todos os tipos de metas flexíveis precisam sobrescrever o método abstrato *getMomentDefuzzify*(). Metas flexíveis decompostas pelo operador *AND* defuzificam retornando o valor do grau de satisfação mais baixo dentre suas metas flexíveis "filhas". Metas flexíveis decompostas pelo operador *OR* defuzificam retornando o valor do grau de satisfação mais alto dentre suas metas flexíveis "filhas". Finalmente, metas flexíveis "folhas" defuzzificam chamando o método *momentDefuzzify*(), oferecido pelo NRC FuzzyJ Toolkit (Orchard 2006).

```
public abstract class Softgoal {
    private String softgoalName;
    private double priority;
    {...}
    public abstract double getMomentDefuzzify();
}
```

Figura 4.10 - Código parcial da classe abstrata "Softgoal"

O algoritmo do simulador de propagação apresentado na Seção 4.1 precisa ser adaptado para servir como uma máquina de raciocínio, adicionando

funcionalidades, tais como acesso às crenças dos agentes, e seguindo as estruturas do JADEX, tais como planos candidatos para atingir uma meta – da classe "ICandidate".

// retorna o plano candidato selecionado
ICandidate simularPropagações (ICandidate[] planos) {
// primeira parte – análise de contexto e inicialização
Para cada plano candidato recuperar a tarefa respectiva do conjunto de crenças "tasks";
Para cada tarefa recuperar do conjunto de crenças "softgoals" as metas flexíveis impactadas;
Para cada tarefa recuperar do conjunto de crenças "contributions" as suas contribuições;
Para cada contribuição criar regras nebulosas de acordo com o tipo de contribuição (como visto na Figura 4.7);

// segunda parte – disparando as regras
Para cada tarefa {
Marcar a tarefa como selecionada, atribuindo "satisficed" ao seu valor nebuloso;
Marcar as outras tarefas como não selecionadas, atribuindo "denied" aos seus valores nebulosos;
Avaliar cada regra de propagação nebulosa e executar as regras compatíveis (método *testRuleMatching*());
Para cada valor nebuloso produzido (os impactos) atribuí-lo à sua respectiva meta flexível;
Salvar as metas flexíveis impactadas como uma alternativa;
}

// terceira parte – avaliação
Para cada alternativa {
Para cada meta flexível {
Defuzificar cada meta flexível (método *getMomentDefuzzify*(), Figura 4.10);

```
      Multiplicar o valor defuzificado pela prioridade da meta flexível e somar o
       resultado ao total da alternativa;
    }
}

 Selecionar o plano candidato associado à tarefa com o maior total de alternativa;
 Notificar os impactos às metas flexíveis a todos os agentes;
 Retornar o plano candidato selecionado;
}
```

É importante destacar que quando um plano é selecionado e executado, os impactos são atribuídos à lista de impactos da meta flexível. Afetam, portanto, o grau de satisfação da meta flexível em tempo de execução. Os impactos atribuídos às metas flexíveis são comunicados a todos os outros agentes da plataforma, na forma de mensagens em broadcast. Os agentes que também utilizam a máquina de raciocínio nebuloso recebem as mensagens e modificam suas crenças de acordo com os impactos. Outros planos, capacidades e mesmo outros agentes podem consultar/checar o grau de satisfação das metas flexíveis a qualquer momento e modificar suas ações de acordo com as novas condições.

Apresentamos cenários de uso da máquina de raciocínio baseada em lógica nebulosa em dois estudos de caso: Lattes-Scholar (Lattes-Scholar 2011a; Lattes-Scholar 2011b) e Sistema de Iluminação Inteligente.

O Lattes-Scholar é uma aplicação *Web* que usa recursos providos por dois serviços *Web*: o bem conhecido Google Scholar e o Lattes (Lattes, 2011). O Lattes é um banco de dados tecnológico e científico mantido pelo Governo Brasileiro. O objetivo do Lattes-Scholar é recuperar o currículo de um pesquisador a partir do Lattes e ordenar as publicações desse pesquisador pelo número de citações, recuperadas usando o Google Scholar. Entretanto, o Google Scholar não aceita várias requisições de uma única máquina (mesmo IP) em um curto período de tempo.

A Figura 4.11 mostra um exemplo de uma estrutura meios-fim do *framework* i* com base no estudo de caso Lattes-Scholar. Essa estrutura é analisada em tempo de execução pela máquina de raciocínio e visa introduzir a variabilidade na forma de como requisitar citações ao Google Scholar. Assim, é

possível requisitar as citações a partir da mesma máquina (mesmo IP) da interface *web* do Lattes-Scholar ou a partir de uma máquina remota, distribuindo as requisições.

O exemplo da Figura 4.11 apresenta uma meta "Número de citações da publicação seja conhecido", duas formas de atingir essa meta, através das tarefas "Requisitar o número de citações para o Google Scholar" e "Delegar a meta a um agente remoto" e dois critérios de qualidade, ou metas flexíveis, que são impactados pela escolha de qual tarefa será executada: "Performance" e "Evitar muitas requisições [Google Scholar]". Quando ambas as metas flexíveis são neutras, como, por exemplo, logo que o Lattes-Scholar é iniciado ou após um período de inatividade, o agente "Fornecedor de Citações" decidirá executar a tarefa ou plano "Requisitar o número de citações para o Google Scholar", uma vez que a meta flexível "Performance" tem prioridade alta. Essa estratégia evita comunicações desnecessárias com agentes remotos.

Entretanto, conforme o grau de satisfação da meta flexível "Evitar muitas requisições [Google Scholar]" cai, o agente passa a selecionar a segunda tarefa, balanceando as requisições entre as máquinas disponíveis.

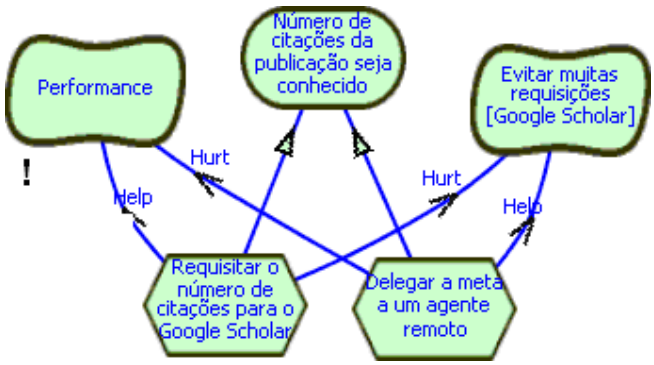


Figura 4.11 - Estrutura meios-fim do Lattes-Scholar

Considerando-se a situação **inicial**, ou seja, ambas as metas flexíveis neutras, podemos descrever passo-a-passo como funciona a máquina de raciocínio nebulosa. A situação inicial das metas flexíveis, isto é, os impactos que cada meta flexível possui antes da análise da máquina de raciocínio, é apresentado na Tabela 4.2. A representação gráfica utilizada para os impactos segue a definição dos conjuntos nebulosos para a variável nebulosa Meta Flexível, Figura 4.1.

A máquina de raciocínio nebulosa analisará as duas alternativas possíveis: (1) selecionar a primeira tarefa para a execução, não executando a segunda tarefa; ou (2) selecionar a segunda tarefa para a execução, não executando a primeira

tarefa. Dessa forma, para a análise da primeira alternativa a máquina de raciocínio atribui impactos às tarefas, de acordo com a Tabela 4.3.

Tabela 4.2 - Situação inicial das metas flexíveis

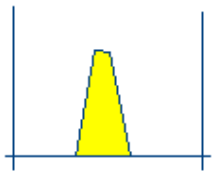
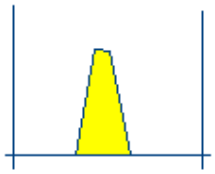
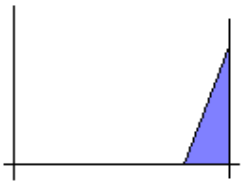
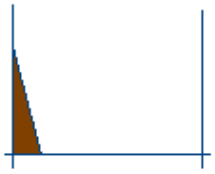
Meta flexível	Impactos
Performance	UNDECIDED, ou 
Evitar muitas requisições [Google Scholar]	UNDECIDED, ou 

Tabela 4.3 - Impactos atribuídos às tarefas na análise da primeira alternativa

Tarefa	Impactos
Requisitar o número de citações para o Google Scholar	SATISFICED, ou 
Delegar a meta a um agente remoto	DENIED, ou 

Após marcar a primeira tarefa como satisfeita e a segunda tarefa como negada, a máquina de raciocínio procede para a execução das regras nebulosas. Somente as regras nebulosas compatíveis com os impactos das tarefas e os tipos de elos de contribuição das tarefas para as metas flexíveis são executadas. Na análise da primeira alternativa, as regras nebulosas executadas são apresentadas na Tabela 4.4.

Tabela 4.4 - Regras nebulosas executadas na análise da primeira alternativa

Regra Nebulosa	Descrição Textual
Regra 15	Se "Requisitar o número de citações para o Google Scholar" é SATISFICED e o elo de contribuição é HELP, então "Performance" é PARTIALLY_SATISFICED.
Regra 14	Se "Requisitar o número de citações para o Google Scholar" é SATISFICED e o elo de contribuição é HURT, então "Evitar muitas requisições [Google Scholar]" é PARTIALLY_DENIED.
Regra 02	Se "Delegar a meta a um agente remoto" é DENIED e o elo de contribuição é HURT, então "Performance" é PARTIALLY_SATISFICED.
Regra 03	Se "Delegar a meta a um agente remoto" é DENIED e o elo de contribuição é HELP, então "Evitar muitas requisições [Google Scholar]" é PARTIALLY_DENIED.

Cada regra nebulosa da Tabela 4.4 atribuirá impactos às metas flexíveis caso essa alternativa seja selecionada. A Regra Nebulosa 15, por exemplo, atribuirá o impacto "PARTIALLY_SATISFICED" à meta flexível "Performance". Os impactos que serão atribuídos às metas flexíveis caso a primeira alternativa seja selecionada são apresentados na Tabela 4.5.

Uma vez analisada a primeira alternativa, a máquina de raciocínio procede para a análise da segunda alternativa, atribuindo impactos às tarefas de acordo com a Tabela 4.6.

Tabela 4.5 - Impactos atribuídos às metas flexíveis caso a primeira alternativa seja selecionada

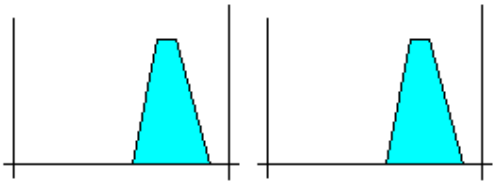
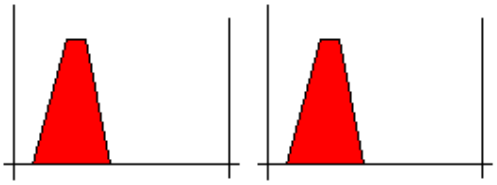
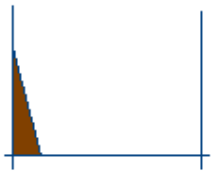
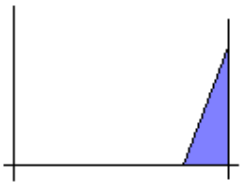
Meta flexível	Impactos
Performance	Dois impactos PARTIALLY_SATISFICED, ou 
Evitar muitas requisições [Google Scholar]	Dois impactos PARTIALLY_DENIED, ou 

Tabela 4.6 - Impactos atribuídos às tarefas na análise da segunda alternativa

Tarefa	Impacto
Requisitar o número de citações para o Google Scholar	DENIED, ou 
Delegar a meta a um agente remoto	SATISFICED, ou 

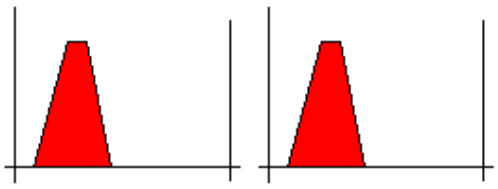
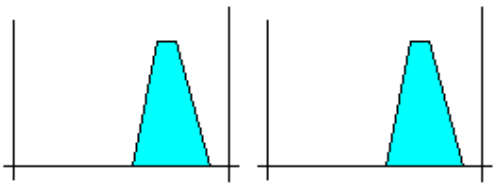
Após marcar a primeira tarefa como negada e a segunda tarefa como satisfeita, a máquina de raciocínio procede para a execução das regras nebulosas. Na análise da segunda alternativa, as regras nebulosas executadas são apresentadas na Tabela 4.7.

Tabela 4.7 - Regras nebulosas executadas na análise da segunda alternativa

Regra Nebulosa	Descrição Textual
Regra 03	Se “Requisitar o número de citações para o Google Scholar” é DENIED e o elo de contribuição é HELP, então “Performance” é PARTIALLY_DENIED.
Regra 02	Se “Requisitar o número de citações para o Google Scholar” é DENIED e o elo de contribuição é HURT, então “Evitar muitas requisições [Google Scholar]” é PARTIALLY_SATISFICED.
Regra 14	Se “Delegar a meta a um agente remoto” é SATISFICED e o elo de contribuição é HURT, então “Performance” é PARTIALLY_DENIED.
Regra 15	Se “Delegar a meta a um agente remoto” é SATISFICED e o elo de contribuição é HELP, então “Evitar muitas requisições [Google Scholar]” é PARTIALLY_SATISFICED.

Os impactos que serão atribuídos às metas flexíveis caso a segunda alternativa seja selecionada são apresentados na Tabela 4.8.

Tabela 4.8 - Impactos atribuídos às metas flexíveis caso a primeira alternativa seja selecionada

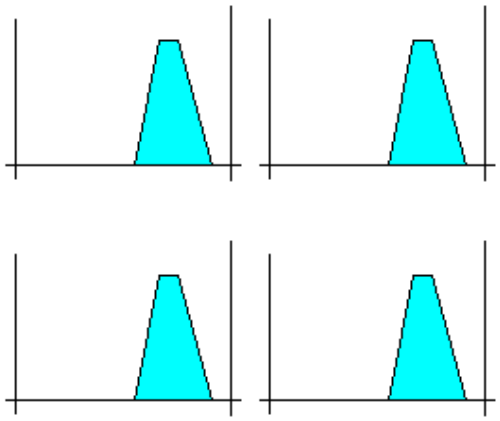
Meta flexível	Impacto
Performance	Dois impactos PARTIALLY_DENIED, ou 
Evitar muitas requisições [Google Scholar]	Dois impactos PARTIALLY_SATISFICED, ou 

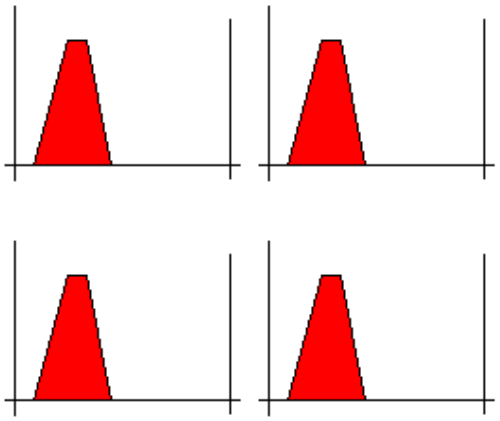
Visto que as duas alternativas são exatamente opostas, ou seja, a primeira alternativa favorece a meta flexível "Performance" e a segunda alternativa favorece a meta flexível "Evitar muitas requisições [Google Scholar]", a máquina de raciocínio nebulosa selecionará a primeira alternativa, uma vez que a "Performance" é uma meta flexível crítica. Assim, o retorno da execução da máquina nebulosa é um objeto contendo o plano candidato respectivo à tarefa "Requisitar o número de citações ao Google Scholar".

A seleção das alternativas é feita pela máquina de raciocínio nebulosa de acordo com o contexto. A seleção da primeira tarefa no exemplo mostrado deve-se à situação inicial das metas flexíveis. Uma segunda execução também levaria à seleção da primeira alternativa. Entretanto, uma terceira execução selecionaria a segunda tarefa, não a primeira.

Para esclarecer como o contexto influencia a seleção realizada pela máquina de raciocínio, detalharemos a seguir a terceira execução da máquina nebulosa. O contexto, ou a situação atual das metas flexíveis, é representado pelos impactos já recebidos pelas metas flexíveis "Performance" e "Evitar muitas requisições [Google Scholar]", apresentados na Tabela 4.9

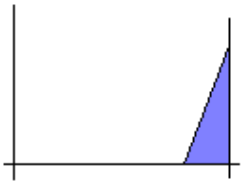
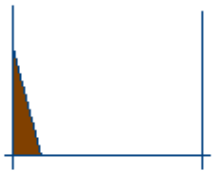
Tabela 4.9 – Situação atual das metas flexíveis

Meta Flexível	Impactos
Performance	Quatro impactos PARTIALLY_SATISFICED, ou 

Evitar muitas requisições [Google Scholar]	Quatro impactos PARTIALLY_DENIED, ou 

A máquina de raciocínio nebulosa analisará as duas alternativas possíveis. Para a análise da primeira alternativa a máquina de raciocínio atribui impactos às tarefas, de acordo com a Tabela 4.10.

Tabela 4.10 - Impactos atribuídos às tarefas na análise da primeira alternativa

Tarefa	Impactos
Requisitar o número de citações para o Google Scholar	SATISFICED, ou 
Delegar a meta a um agente remoto	DENIED, ou 

Após marcar a primeira tarefa como satisfeita e a segunda tarefa como negada, a máquina de raciocínio procede para a execução das regras nebulosas. As

regras nebulosas executadas na análise da primeira alternativa são apresentadas na Tabela 4.11.

Tabela 4.11 - Regras nebulosas executadas na análise da primeira alternativa

Regra Nebulosa	Descrição Textual
Regra 15	Se "Requisitar o número de citações para o Google Scholar" é SATISFICED e o elo de contribuição é HELP, então "Performance" é PARTIALLY_SATISFICED.
Regra 14	Se "Requisitar o número de citações para o Google Scholar" é SATISFICED e o elo de contribuição é HURT, então "Evitar muitas requisições [Google Scholar]" é PARTIALLY_DENIED.
Regra 02	Se "Delegar a meta a um agente remoto" é DENIED e o elo de contribuição é HURT, então "Performance" é PARTIALLY_SATISFICED.
Regra 03	Se "Delegar a meta a um agente remoto" é DENIED e o elo de contribuição é HELP, então "Evitar muitas requisições [Google Scholar]" é PARTIALLY_DENIED.

Os quatro impactos que serão atribuídos às metas flexíveis caso a primeira alternativa seja selecionada são adicionados aos oito impactos iniciais, formando o conjunto de impactos resultante apresentado na Tabela 4.12.

O resultado da análise da primeira alternativa mostra que a meta flexível "Evitar muitas requisições [Google Scholar]" se aproxima de ser negada.Uma vez analisada a primeira alternativa, a máquina de raciocínio procede para a análise da segunda alternativa, atribuindo impactos às tarefas de acordo com a Tabela 4.13.

Tabela 4.12 – Conjunto de impactos resultante da seleção da primeira alternativa

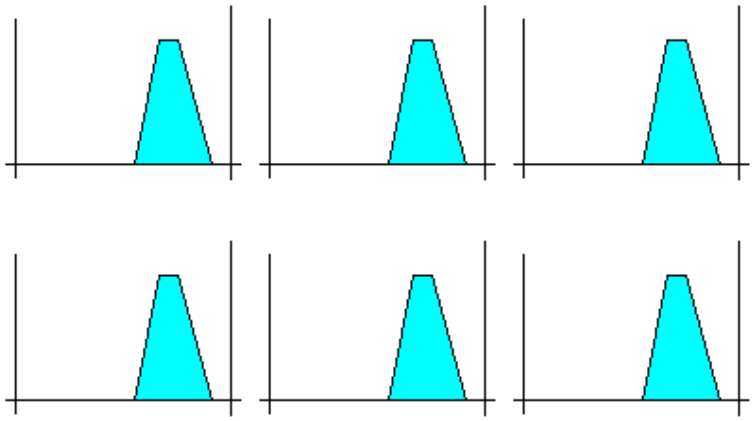
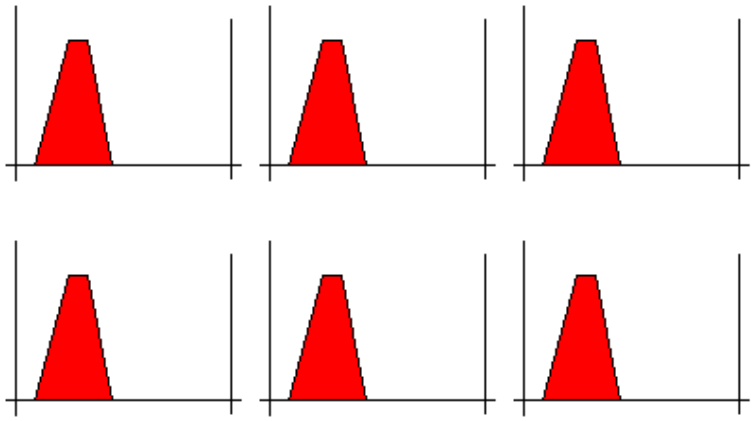
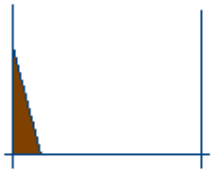
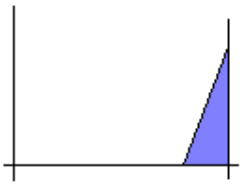
Meta flexível	Impactos
Performance	Seis impactos PARTIALLY_SATISFICED, ou 
Evitar muitas requisições [Google Scholar]	Seis impactos PARTIALLY_DENIED, ou 

Tabela 4.13 - Impactos atribuídos às tarefas na análise da segunda alternativa

Tarefa	Impacto
Requisitar o número de citações para o Google Scholar	DENIED, ou 

Delegar a meta a um agente remoto	SATISFICED, ou 

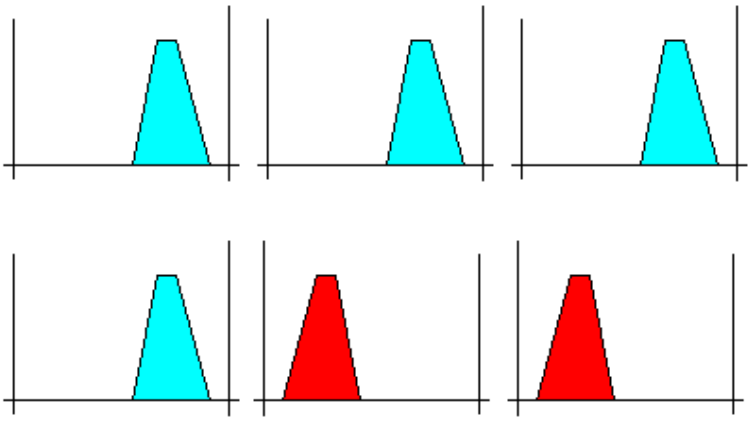
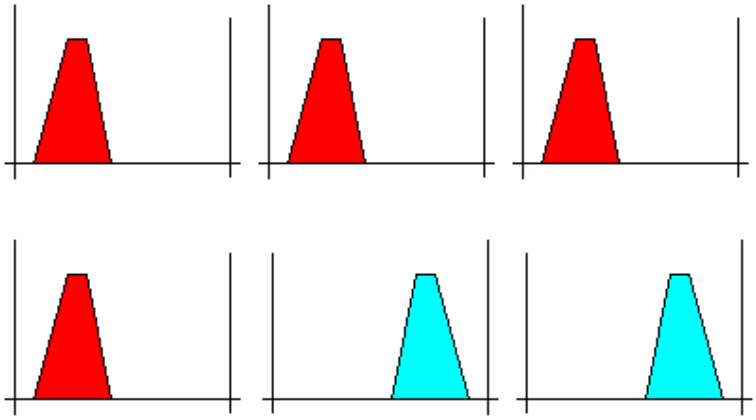
Após marcar a primeira tarefa como negada e a segunda tarefa como satisfeita, a máquina de raciocínio procede para a execução das regras nebulosas. As regras nebulosas executadas na análise da segunda alternativa são apresentadas na Tabela 4.14.

Tabela 4.14 - Regras nebulosas executadas na análise da segunda alternativa

Regra Nebulosa	Descrição Textual
Regra 03	Se "Requisitar o número de citações para o Google Scholar" é DENIED e o elo de contribuição é HELP, então "Performance" é PARTIALLY_DENIED.
Regra 02	Se "Requisitar o número de citações para o Google Scholar" é DENIED e o elo de contribuição é HURT, então "Evitar muitas requisições [Google Scholar]" é PARTIALLY_SATISFICED.
Regra 14	Se "Delegar a meta a um agente remoto" é SATISFICED e o elo de contribuição é HURT, então "Performance" é PARTIALLY_DENIED.
Regra 15	Se "Delegar a meta a um agente remoto" é SATISFICED e o elo de contribuição é HELP, então "Evitar muitas requisições [Google Scholar]" é PARTIALLY_SATISFICED.

Os quatro impactos que serão atribuídos às metas flexíveis caso a segunda alternativa seja selecionada são adicionados aos oito impactos iniciais, formando o conjunto de impactos resultante apresentado na Tabela 4.15.

Tabela 4.15 – Conjunto de impactos resultante da seleção da segunda alternativa

Meta flexível	Impacto
Performance	Quatro impactos PARTIALLY_ SATISFICED e dois impactos PARTIALLY_ DENIED, ou 
Evitar muitas requisições [Google Scholar]	Quatro impactos PARTIALLY_ DENIED e dois impactos PARTIALLY_SATISFICED, ou 

Visto que em ambas as alternativas a meta flexível "Performance", que é a meta flexível crítica, continua sendo parcialmente satisfeita, a máquina de raciocínio nebulosa selecionará a segunda alternativa, uma vez que a meta flexível "Evitar muitas requisições [Google Scholar]" passa a ter um valor um pouco menos negativo, se afastando de ser negada.

Apresentaremos a seguir um segundo exemplo, O Sistema de Iluminação Inteligente (SII), porém com menos detalhes. O SII é um sistema multi-agentes

para controlar a iluminação de um edifício. Nesse contexto, alguns critérios de qualidade são inerentes, tais como segurança, privacidade e consumo de energia. Além disso, cada ação afeta as metas flexíveis relacionadas. Por exemplo, apagar a luz pode causar um risco para a segurança, caso não exista luz natural no ambiente.

A Figura 4.12 ilustra um exemplo de uma estrutura meios-fim do *framework* i* com base no estudo de caso Sistema de Iluminação Inteligente. Essa estrutura é analisada em tempo de execução pela máquina de raciocínio.

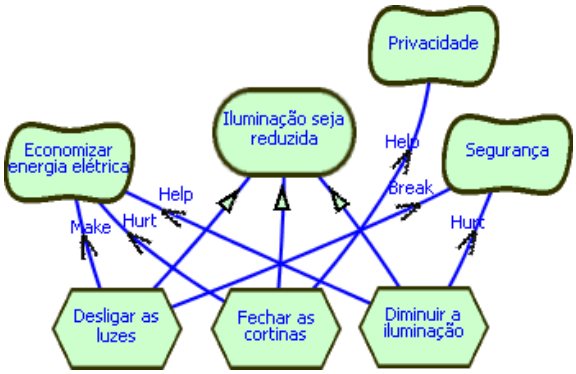


Figura 4.12 - Estrutura meios-fim do Sistema de Iluminação Inteligente

Quando todas as metas flexíveis forem neutras, o agente decidirá diminuir a iluminação. Entretanto, caso a meta flexível "Privacidade" tenha um baixo grau de satisfação, o agente executará a tarefa ou plano "Fechar as cortinas". O agente evitará deixar qualquer meta flexível assumir um grau de satisfação negativo. Note que outros fatores (estruturas meios-fim) afetam as metas flexíveis. Por exemplo, é impossível que a meta flexível "Segurança" seja negada (*breaked*) na presença de luz natural.

4.3. Trabalhos Relacionados

Nessa seção, comparamos nossa proposta com as categorizações propostas pelos trabalhos (Supakkul et al. 2010) e (Horkoff e Yu 2011) para diferentes técnicas comumente encontradas na literatura. Adicionalmente, apresentamos alguns trabalhos relacionados, classificando-os quanto às categorias propostas.

De acordo com (Supakkul et al. 2010), os autores descrevem dois esquemas de seleção como padrões: padrão de seleção quantitativo baseado em peso e padrão de seleção qualitativo baseado em *rank*. O primeiro padrão está

centrado em esquemas de seleção comumente (Van Lamsweerde 2009; Xie et al. 2009) usados na seleção de alternativas de um nível em modelos orientados a metas. Apesar da nossa abordagem analisar alternativas de um nível, ao se aplicar lógica nebulosa garante-se uma seleção qualitativa. O ultimo padrão é centrado em esquemas de seleção qualitativos para escolher alternativas em modelos orientados a metas (Elahi e Yu 2009). Um exemplo de esquema de seleção é a bem conhecida seleção por pares. Esse tipo de esquema de seleção é útil quando se deseja lidar com conhecimento específico sobre as alternativas. É relevante mencionar que esse conhecimento precisa estar disponível. Supakkul et al. aplicam esse padrão para sistematicamente selecionar uma alternativa considerando um esquema de *ranking*, que deve estar de acordo com os desejos dos interessados. Esse mecanismo permite enumerar preferências para cada tipo de contribuição com base em diferentes prioridades. Optamos por oferecer uma abordagem simples para lidar com todas as metas flexíveis. Customizar cada análise por meta flexível pode levar a um comportamento de difícil solução – i.e. um comportamento errático, ou seja, difícil de se corrigir – uma vez que nossa análise ocorre em tempo de execução. Além disso, não permitimos customização por tipo de contribuição. Lidamos com tipos de contribuição seguindo as regras de propagação da Tabela 4.1, pois nosso foco está na simulação das práticas dos engenheiros de requisitos durante o processo de análise.

Em (Horkoff e Yu 2011), os autores provêem um *survey* de modelos orientados a metas e também sugerem um guia inicial na escolha de técnicas para conhecer/encontrar/identificar os desejos dos interessados. O foco do *survey* está em técnicas que analisam um modelo depois da sua criação ao invés de em técnicas que direcionam a criação de modelos. Nesse contexto, Horkoff and Yu apresentam vários procedimentos, agrupando os mesmos em categorias de acordo com as técnicas utilizadas: Análise de Satisfação, Métricas, Planejamento, Simulação e Checagem de Modelos. Com base nos autores:

i. os procedimentos de Análise de Satisfação começam com valores iniciais atribuídos ao modelo e então usam elos do modelo para propagar valores em ambos os sentidos: avante (i.e. *forward* (Amyot et al. 2010; Chung et al. 2000)) ou reverso (i.e. *backward* (Giorgini et al. 2004, Horkoff e Yu 2010)). Além disso, alguns procedimentos de análise (Chung et al. 2000; Amyot et al. 2010; Horkoff e Yu 2010) provêem resultados centrados em etiquetas qualitativas, tais

como: *satisfied*, *partially satisfied*, *conflict*, *unknown*, *partially denied*, e *denied*. Outros procedimentos de análise (Maiden et al. 2007) provêem resultados binários através da atribuição de um único valor dentre dois possíveis (e.g. *satisfied* ou *not satisfied*). Existem vários procedimentos que provêem análise quantitativa centrada em números para indicar, por exemplo, o grau de satisfação (Amyot et al. 2010). Com base no *survey* dos autores Horkoff e Yu, é relevante notar que nossa abordagem propaga no sentido avante (i.e. *forward*). Aplicamos a lógica nebulosa exatamente para lidar com etiquetas qualitativas em tempo de execução. Como pode ser observado na Figura 4.7, nossas regras nebulosas não são binárias. Finalmente, a análise por lógica nebulosa está centrada em qualitativamente indicar o grau de satisfação;

ii. os procedimentos de Métricas pretendem "medir" critérios de qualidade no domínio em questão. Os autores apresentam alguns trabalhos relacionados nesse sentido, tais como (Franch e Maiden 2003; Franch et al. 2004). No primeiro trabalho, Franch e Maiden oferecem uma lista de classificações de dependência em um modelo SD como parte de fórmulas quantitativas visando obter valores para alguns critérios de qualidade, tais como: vulnerabilidade e uniformidade. No último trabalho, Franch et al. incorporaram à abordagem deles os meios para obter métricas globais e locais em modelos SD – vide Capítulo 2: Seção 2.1 – centrados em classificações e pesos dos autores e dependências. Nossa abordagem não pode ser considerada um procedimento de métrica, uma vez que analisamos de forma qualitativa o grau de satisfação das metas flexíveis;

iii. os procedimentos de Planejamento têm utilizado abordagens de planejamento da área de Inteligência Artificial para obter uma sequencia satisfatória de tarefas ou estratégias de projeto em modelos orientados à meta. Os autores detalharam esses procedimentos apresentando algumas contribuições de outros autores, tais como (Bryl et al. 2006; Asnar et al. 2007). Na primeira contribuição, Bryl et al. tentam oferecer um suporte para interativamente encontrar delegações satisfatórias – i.e. atribuições de dependências – em um modelo orientado à meta, obtendo dessa forma planos que satisfazem completamente todos os atores. Baseados nos planos obtidos, Bryl et al. propõem avaliar esses planos levando em consideração o custo. Na última contribuição, Asnar et al. introduzem análise de risco em (Asnar et al. 2006) para medir o nível mínimo de confiança necessário para uma delegação entre atores. Intervenções

dos projetistas também são um recurso para refinar o planejamento. Nossa abordagem não é um procedimento de Planejamento, uma vez que não está focada na determinação de uma sequencia de tarefas visando atingir às metas. Entretanto, nossa abordagem pode ajudar o software no balanceamento do grau de satisfação das metas flexíveis, evitando, portanto, riscos desnecessários;

iv. os procedimentos de Simulação são caracterizados – de acordo com (Horkoff e Yu 2011) – por adicionar informações temporais em modelos orientados à meta. Eles permitem, por exemplo, simular construções de modelo com o intuito de checar/verificar propriedades inesperadas ou indesejadas em um cenário em particular. Os autores apresentam alguns procedimentos de Simulação, tais como (Gans et al. 2003). Esse exemplo de procedimento estende modelos de metas i* com informação temporal, modificando partes desses modelos e excluindo metas flexíveis e contribuições. Dessa forma, o comportamento dos atores é simulado também seguindo a invocação/chamada de ações e usando interações com os cidadãos. Apesar da nossa abordagem simular regras de propagação, ela não foca em simulação de modelos orientados à meta. Entretanto, simulamos modelos orientados à meta para ajustar pequenos detalhes – i.e. ajuste fino – nos nossos conjuntos nebulosos. Essa estratégia pode ser realizada criando agentes de software do tipo "casca", isto é, sem uma implementação de planos completa; e

v. os procedimentos de Checagem de Modelo visam realizar verificações em modelos orientados à meta. Nesse caso, os autores apresentam duas abordagens específicas (Bryl et al. 2007; Gans et al. 2004). Ambas as abordagens aplicam a combinação de Checagem de Modelo com outros dois procedimentos previamente mencionados, Simulação e Planejamento. Além disso, eles descrevem o trabalho proposto em (Bryl et al. 2006), no qual Bryl et al. lidam com restrições de privacidade e segurança, considerando-as no planejamento. A principal ideia é derivar automaticamente e selecionar alternativas de projeto durante o processo de desenvolvimento do software, visando obter um sistema seguro. Nossa abordagem não é um procedimento de Checagem de Modelo, uma vez que a mesma foca na análise de alternativas em tempo de execução.

Como previamente apresentado, existem diferentes técnicas quantitativas e qualitativas para realizar seleção de alternativas. Na nossa abordagem, usamos uma das técnicas mais comumente aplicadas, centrada em regras de propagação,

observando as práticas dos engenheiros de requisitos. Adicionalmente, de acordo com a categorização proposta por Horkoff e Yu, nossa proposta é uma abordagem híbrida, combinando principalmente procedimentos qualitativos de Análise de Satisfação com a técnica de Simulação. Concluindo, nossa máquina de raciocínio não simula todo o modelo de metas, mas simula os efeitos relacionados à seleção de cada alternativa para atingir as metas especificadas.

4.4. Considerações Finais

Nesse capítulo, apresentamos um simulador de propagação baseado em lógica nebulosa que replica, em tempo de execução, a análise praticada pelos engenheiros de requisitos quando aplicam regras de propagação. Contribuímos ainda com uma implementação em Java desse simulador de propagação na forma de uma máquina de raciocínio para agentes intencionais. Essa máquina de raciocínio permite analisar qualitativamente os graus de satisfação das metas flexíveis, representando as expressões lingüísticas qualitativas (como *denied*, *satisficed*, *partially_satisficed*, entre outras) como conjuntos nebulosos. Os agentes tomam decisões levando em consideração o contexto em tempo de execução, e então selecionam um plano adequado para atingir as metas desejadas. O contexto avaliado inclui os graus de satisfação das metas flexíveis, o conhecimento dos agentes sobre o modelo i* e os planos candidatos.

Oferecemos a nossa máquina de raciocínio como uma capacidade JADEX (Braubach et al. 2005). Essa capacidade pode facilmente ser reutilizada por qualquer agente intencional. A implementação em Java da máquina de raciocínio também pode ser facilmente integrada com ferramentas CASE baseadas em Java.