2.
Conceitos e Apoio Tecnológico

Neste capítulo apresentamos brevemente alguns conceitos e apoios tecnológicos que servem como base para a nossa abordagem. Embora outros também possam ser listados, a compreensão dos quatros conceitos ou apoios tecnológicos aqui descritos é fundamental para o pleno entendimento da nossa abordagem visando o desenvolvimento de software transparente.

Esse capítulo está organizado da seguinte forma: a Seção 2.1 apresenta dois *frameworks* intencionais para a modelagem de requisitos, os *frameworks* i* e NFR. A Transparência de Software bem como o Catálogo de Transparência de Software são discutidos na Seção 2.2. A Seção 2.3 introduz o apoio tecnológico oferecido pelos sistemas multi-agentes intencionais. O *framework* de argumentação *Acceptability Evaluation* (ACE) e seu algoritmo de análise de grafos de argumentação são explicados na Seção 2.4. Finalmente, a Seção 2.5 apresenta as considerações finais desse capítulo.

2.1.
Modelos Intencionais: Os *Framework*s i* e NFR

O *framework* i* foi proposto por Eric Yu em sua tese de doutorado (Yu 1995) em 1995. Sua tese possibilitou a publicação de uma série de artigos na área de Engenharia de Requisitos. Dentre esses artigos, um que merece especial menção é o (Yu 1997), que gerou grande reconhecimento na comunidade científica, publicado no 3rd IEEE International Conference on Requirements Engineering (RE), IEEE Computer Society. Após dez anos, as contribuições da tese (Yu 1995) foram consideradas no RE'08 como um marco na área de Engenharia de Requisitos, contribuindo não somente, mas em especial, para a modelagem intencional de software em contextos organizacionais. Por definição, i* significa intencionalidade distribuída.

Essa proposta de modelo intencional (Yu 1995; Yu 1997; ISTARWIKI 2011) oferece um conjunto de conceitos – i.e. abstrações, também detalhadas em

(Yu 1995) na linguagem Telos (Kramer et al. 1991) – para modelar as intenções dos interessados em um software que atua em contextos organizacionais. Dentre as principais abstrações do i*, têm-se: (i) atores – que representam entidades ativas que podem executar ações para atingir metas e dependem de outros atores; (ii) metas – que representam os objetivos dos atores no que tange às funcionalidades do software em desenvolvimento; (iii) tarefas – que representam ações que os atores podem realizar no ambiente visando atingir suas metas, e (iv) metas flexíveis – que representam qualidades desejáveis que não possuem critérios claros para sua satisfação, ou seja, são subjetivas e dependentes dos pontos de vista dos interessados. O *framework* i* também inclui vários elos de associações entre atores i* (tais como: é_parte_de, é_um, desempenha e ocupa), dependências estratégicas entre atores e elos meio-fim que relacionam as meta (o fim) às ações (os meios) que viabilizam atingi-la. (Oliveira 2008) propõe uma extensão para o conceito dos modelos SD e SR visando separá-los por situação.

A Figura 2.1 ilustra um exemplo simples de um modelo i* sobre o pagamento de uma conta elétrica que consiste nos atores "Consumidor" e "Banco", onde o ator "Consumidor" deseja atingir a meta "Conta de energia elétrica seja paga" e existem duas tarefas (meios) para atingir essa meta: "Pagar a conta de energia elétrica em dinheiro" e "Pagar a conta de energia elétrica com o cartão de crédito". A primeira tarefa contribui positivamente (elo Help) para satisfazer a meta flexível "Privacidade". A segunda tarefa contribui positivamente (elo Help) para satisfazer a meta flexível "Conveniência". O ator "Banco", por outro lado, deseja atingir a meta "Obter novos clientes" e tem o meio "Oferecer um cartão de crédito ao cliente" para atingi-la. Essa tarefa contribui positivamente (elo Help) para satisfazer a meta flexível "Maximizar o lucro".

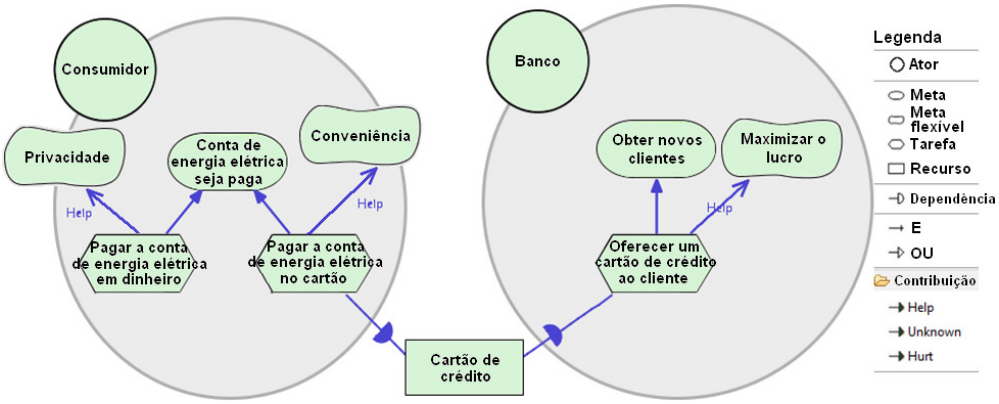

Figura 2.1 - Um modelo i*

O NFR Framework foi proposto por Chung et al. (Chung et al. 2000) no intuito de modelar requisitos não-funcionais usando grafos conhecidos como *Softgoals Interdependency Graphs* (SIGs). Esse livro relata um dos primeiros esforços da comunidade científica no sentido de lidar com requisitos não-funcionais, tipicamente subjetivos e comumente esquecidos – até aquele momento – no processo de desenvolvimento de software.

O foco do modelo intencional proposto pelo NFR Framework são os requisitos não-funcionais e suas operacionalizações, e não a distribuição desses conceitos através de uma rede de atores, como proposto no modelo intencional do i*. Dentre as principais abstrações usadas na modelagem intencional do NFR Framework, destacam-se: (i) metas flexíveis – que representam critérios de qualidade com a mesma semântica das metas flexíveis dos modelos i*, (ii) operacionalizações – que representam diferentes formas de operacionalizar parcialmente uma meta flexível, e (iii) argumentos – que representam afirmações que servem como notas em um SIG e oferecem o *rationale* do desenho. O *framework* também oferece elos de contribuição e correlação para indicar influências entre metas flexíveis, operacionalizações e argumentos. Adicionalmente, as metas flexíveis podem ser decompostas em outras metas flexíveis para refinar e detalhar a modelagem de um requisito não-funcional. A Figura 2.2 mostra um exemplo simples de um SIG, detalhando o requisito não-funcional "Rastreabilidade [Dados]".

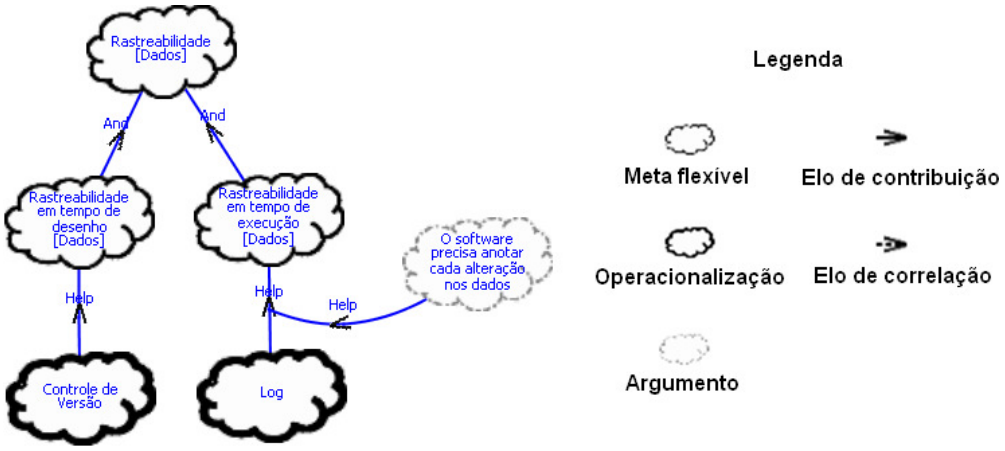

Figura 2.2 - Um exemplo de um SIG

Na Figura 2.2, a meta flexível "Rastreabilidade [Dados]" é decomposta – decomposição representada pelo elo And – nas metas flexíveis "Rastreabilidade

em tempo de desenho [Dados]" e "Rastreabilidade em tempo de execução". Além disso, a operacionalização "Controle de Versão" contribui positivamente – contribuição representada pelo elo Help – para a meta flexível "Rastreabilidade em tempo de desenho [Dados]". No caso da operacionalização "Log", a mesma contribui positivamente para a meta flexível "Rastreabilidade em tempo de execução [Dados]". Já o argumento "The software must log each data update" justifica a contribuição positiva da operacionalização "Log".

2.2. Transparência de Software

Leite e Cappelli (Leite e Cappelli 2010) tratam a transparência de software como um requisito não-funcional e oferecem um SIG que relaciona a meta flexível Transparência com Acessibilidade, Usabilidade, Informatividade, Entendimento e Auditabilidade. A Figura 2.3 mostra o SIG de Transparência, uma tradução do SIG proposto em (Leite e Cappelli 2010; Aló 2009).

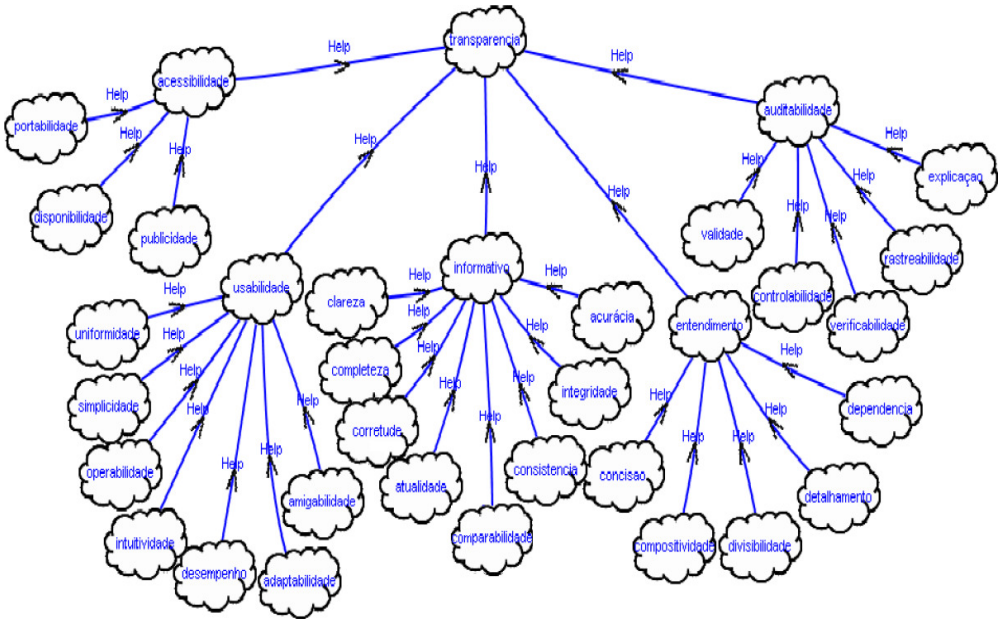

Figura 2.3 - O SIG de Transparência, traduzido de (Leite e Cappelli 2010)

Em (CTS 2011), o Catálogo de Transparência de Software foi refinado pelos passos 1 e 2 do método GQM - *Goal*, *Question*, *Metric* (Basili 1992). Cada folha do SIG de Transparência, por exemplo, Validade, foi refinada ao quarto e quinto níveis de detalhe, como mostrado na Tabela 1.

Tabela 2.1 - Operacionalizações e questões para a meta flexível folha Validade

Op1	Identificar os mecanismos de validação
Q1.1	Foram identificados meios de se comprovar as condições iniciais?
Q1.2	Foram identificados meios de se comprovar as condições finais?
Q1.3	Foram identificados meios de se comprovar os requisitos funcionais?
Q1.4	Foram identificados meios de se comprovar os requisitos não-funcionais?
Q1.5	As fontes de informação foram identificadas?
Op2	Aplicar um mecanismo de validação nos artefatos
	...

Se considerarmos Validade como uma meta mensurável, então as linhas cinzas são suas operacionalizações e as linhas brancas são questões. As respostas para essas questões constituem formas de se implementar cada operacionalização do SIG. Portanto, as mesmas compõem um catálogo das formas possíveis de se atingir uma propriedade qualitativa desejável.

2.3. Sistemas Multi-Agentes Intencionais

De acordo com (Wooldridge 2002), um sistema multi-agentes (SMA) é um sistema composto de vários agentes inteligentes que trabalham de forma colaborativa para atingir seus objetivos. Em outras palavras, esses agentes inteligentes tendem a encontrar automaticamente a melhor solução, isto é, encontram-na sem intervenção externa. Portanto, os agentes podem assumir diferentes responsabilidades – normalmente representadas como diferentes papéis ou capacidades. Além disso, eles podem executar diversas tarefas, adaptando-se para atingir os objetivos.

Um sistema multi-agentes pode ser dirigido por agentes comportamentais (Bellifemine et al. 2007) ou agentes intencionais (Bratman 1999). De um lado, a primeira orientação é mais apropriada para lidar com sistemas baseados em agentes que podem ser modelados e implementados utilizando-se comportamentos específicos, como comer, dormir, correr, ler, entre outros. Portanto, a orientação baseada na abstração de comportamento parece adequada para representar, por exemplo, auto-organização em colônias sociais (como, por exemplo, colônias de formigas). Por outro lado, a orientação baseada na abstração de intencionalidade pode melhorar o raciocínio e a capacidade de aprendizado de agentes inteligentes usando conceitos como crença, desejo e intenção.

De acordo com (Braubach et al. 2003), um sistema multi-agentes intencional representa uma forma adequada de lidar com o raciocínio prático humano, simular a formação de metas, e interpretar os estados mentais humanos. O modelo *Belief-Desire-Intention* (BDI) (Bratman 1999) é um modelo intencional intensamente investigado na área da Inteligência Artificial. Trata-se de um modelo proposto por Bratman como uma forma de se representar o raciocínio humano de forma prática. Esse modelo representa nossas atitudes mentais como crenças – o que sabemos; desejos – o que queremos; e intenções – o que podemos fazer. Mais recentemente, esse modelo foi aplicado em agentes inteligentes. Assim, é possível desenvolver sistemas que se utilizam de agentes inteligentes, possibilitando a simulação do raciocínio humano e permitindo lidar com (i) o conhecimento do agente sobre, por exemplo, o mundo real ou um contexto específico; (ii) as metas delegadas aos agentes e a serem atingidas pelos mesmos; e (iii) sequencias de tarefas que devem ser executadas pelo agente com o propósito de atingir as metas especificadas.

O conhecimento do agente é representado pelo conceito de crença. As crenças de um agente são abstrações de informações e de objetos do mundo real, e representam como o agente percebe o ambiente em sua volta. O agente não precisa ter uma descrição completa dessas informações ou objetos; basta uma descrição parcial que enfatize os aspectos relevantes para seu processo de decisão. Uma descrição do estado interno do agente também pode complementar suas crenças.

As metas delegadas aos agentes são representadas pelo conceito de desejo. Os desejos de um agente são estados internos (do próprio agente) ou externos (do ambiente) que o agente pretende alcançar ou manter. Esses desejos são definidos como metas que o agente deve perseguir. Portanto, os desejos são as razões por trás das ações do agente. A principal vantagem dos agentes possuírem metas é que se evita que os agentes sejam meramente reativos. Assim, os agentes adquirem a capacidade de serem também pró-ativos.

As sequências de tarefas que devem ser executadas pelo agente são representadas pelo conceito de intenção. As intenções de um agente representam as ações que o agente pode executar em seu ambiente, de forma a alterá-lo. Um agente pode possuir um conjunto diverso de ações para modificar o ambiente visando satisfazer um único desejo.

Um agente inteligente que implementa o modelo BDI possui uma máquina de deliberação de reações que observa os desejos ativos do agente. Essa máquina analisa as crenças de como o estado interno do agente ou o estado do ambiente se encontram. Além disso, essa máquina analisa as crenças de como o agente deseja que seu estado interno ou o ambiente sejam. Baseado nas diferenças entre esses dois conjuntos de crenças, a máquina de deliberação de reação escolhe as intenções que devem ser executadas, sendo que o suposto impacto dessas intenções é conhecido *a priori*.

No intuito de auxiliar o desenvolvimento de software intencional dirigido por agentes inteligentes, implementados com base no modelo BDI, utilizamos um *framework* específico, o *framework* JADEX (Braubach et al. 2003). Esse *framework* foi proposto por um grupo de pesquisa da Hamburg University. O JADEX, ou "JADE eXtension", é um *add-on* para a plataforma multi-agentes JADE que implementa o modelo BDI. JADEX possibilita o desenvolvimento de agentes de arquitetura híbrida reativa/deliberativa, adicionando o conceito de atitudes mentais informativas (crenças), motivacionais (desejos) e deliberativas (ações ou planos) a agentes da plataforma JADE. A plataforma JADE é uma plataforma multi-agentes conhecida, largamente utilizada, que segue o protocolo FIPA - um padrão de fato para comunicação entre agentes.

O uso do *framework* JADEX permite que artefatos de desenho baseados no modelo BDI possam ser implementados na plataforma JADE seguindo o mesmo paradigma, o que garante uma transição suave entre os níveis de abstração (Braubach et al. 2003). Assim, é possível desenvolver agentes para a plataforma JADE seguindo o paradigma BDI que simula o raciocínio humano, ao invés de termos que desenvolver agentes seguindo o paradigma comportamental, que surgiu de trabalhos com inteligência artificial para robôs móveis (Brooks 1991) e simula sistemas biológicos simples, como insetos.

A Figura 2.4, extraída de (Pokahr e Braubach 2007), apresenta uma visão geral da arquitetura intra-agente utilizada no JADEX. Essa arquitetura se baseia em uma máquina de deliberação de reações que analisa o estado das crenças, adota novas metas, instancia e seleciona planos para execução, enquanto lida com eventos internos e mensagens externas.

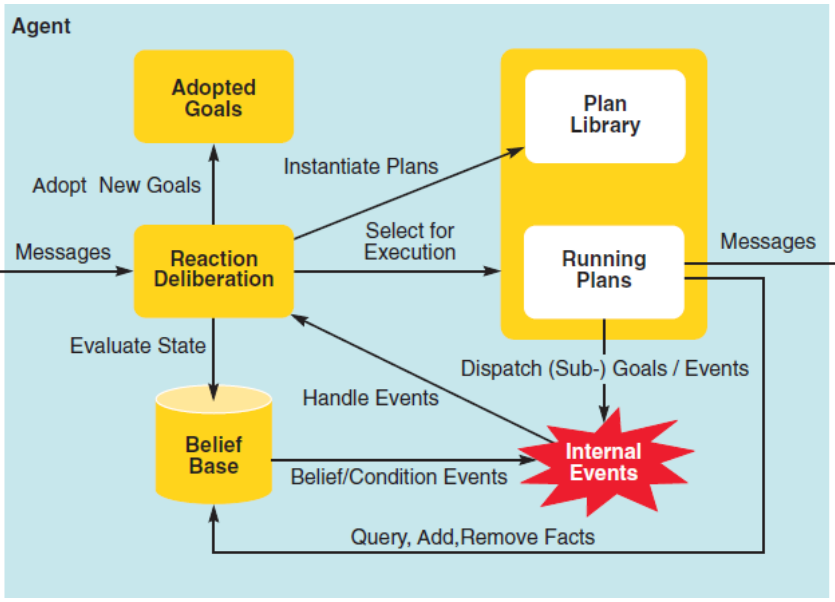

Figura 2.4 - Visão Geral da Arquitetura intra-agente utilizada no JADEX, extraída de (Pokahr e Braubach 2007)

Um agente do *framework* JADEX é definido em um arquivo XML denominado *Agent Definition File* (ADF), onde são declaradas as crenças, metas e planos dos agentes, além de capacidades, mensagens, eventos internos, configurações, entre outros. A Figura 2.5 apresenta o schema XML de um agente JADEX, extraído de (JADEX 2007a).

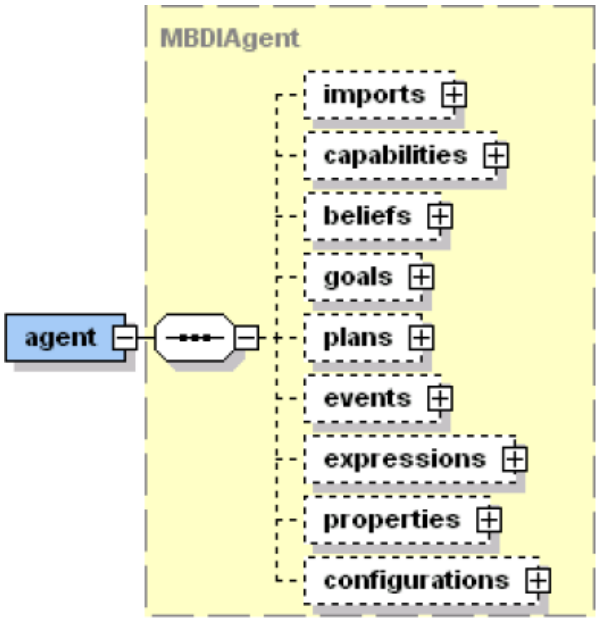

Figura 2.5 - O schema XML de um agente Jadex, extraído de (JADEX 2007a)

Com base no modelo BDI tradicional, proposto por Bratman, as crenças no modelo BDI do JADEX também representam o conhecimento do agente com

base no contexto em análise, ou mesmo em informações do mundo real. As metas continuam representando os objetivos delegados aos agentes com base nos desejos dos interessados. Já os planos representam as sequências de ações do modelo BDI tradicional que são executadas pelos agentes intencionais no intuito de atingir as metas desejadas.

As crenças e os planos são implementados seguindo o paradigma da orientação a objetos na linguagem Java, a linguagem padrão para se implementar agentes para a plataforma JADE. Dessa forma, por exemplo, cada plano é uma classe Java. Vários arquivos Java são armazenados em uma biblioteca de planos. A máquina de raciocínio do JADEX oferece recursos para que – em tempo de execução – um agente tenha condições de selecionar um plano específico da biblioteca de planos visando atender seus objetivos. Dentre outras contribuições, esses recursos viabilizam a análise do contexto em tempo de execução – i.e. condições atuais do ambiente organizacional, tais como preferências dos interessados e possíveis conflitos entre os interessados – melhorando a capacidade cognitiva do agente intencional e, conseqüentemente, sua tomada de decisão através da escolha de um plano adequado, que atenda melhor as condições impostas. A Figura 2.6 ilustra um plano, extraído de (JADEX 2007a), que escolhe um movimento entre vários possíveis de um agente que participa de um jogo.

```
public void body()
{
    ICandidateInfo[] apps = (ICandidateInfo[])getParameterSet("applicables").getValues();

    ICandidateInfo sel = null;

    for(int i=0; i<apps.length; i++)
    {
        // Decide which plan to select, e.g. using the move parameter of the move_plan.
        Move move = (Move)apps[i].getPlan().getParameter("move").getValue();
        ...
    }

    getParameterSet("result").addValue(sel);
}
```

Figura 2.6 - Exemplo de um plano que escolhe um movimento entre vários possíveis, extraído de (JADEX 2007a)

2.4. O *Framework Acceptability Evaluation*

O *framework Acceptability Evaluation* (ACE) (Jureta et al. 2009) oferece meios para se modelar e pensar na validade relativa de requisitos elicitados. A validade relativa significa que os interessados concordam com o conteúdo de um

artefato da Engenharia de Requisitos (doravante ER), ou seja, requisitos elicitados, seus refinamentos, e informações assumidas sobre eles. Esse *framework* consiste em (i) um conjunto de conceitos para modelar os argumentos dos interessados como um grafo, e (ii) um algoritmo para avaliar a aceitabilidade do grafo (para os interessados). Adicionalmente, os autores propõem a condição de aceitabilidade, para sinalizar o fim de uma discussão.

ACE modela os artefatos da ER e os argumentos dos interessados como proposições, seguindo McGrath (McGrath 2008) apud (Jureta et al. 2009). Modelos de argumentos são grafos direcionados e etiquetados, onde todos os elos possuem a etiqueta implícita "para" e os nós representam as proposições com quatro possíveis etiquetas: **i**, **I**, **C** e **P**. Nós etiquetados como **I** são nós de inferência, um meio ampliativo ou dedutivo para obter as saídas (fins) a partir das entradas. Nós etiquetados como **i** são vértices de informação, usados como entradas ou saídas de regras de inferência. Nós etiquetados como **C** são vértices de conflito, anotando conflitos entre outros vértices do grafo. Finalmente, nós **P** representam preferências entre dois outros vértices.

A Figura 2.7 mostra um exemplo do ACE. As proposições *p1*, *p2* e *p3* são respectivamente representadas por vértices de informação i(*p1*), i(*p2*) e i(*p3*). A partir da proposição *p1*, podemos inferir a proposição *p2*, de acordo com o nó I_T. Esta regra de inferência pode ser formalmente descrita como I_T(i(*p1*),i(*p2*)). A proposição *p3* conflita com a proposição *p2*, representada pelo nó C1 (C1(i(*p3*),i(*p2*))). P1 representa uma preferência por *p2* em relação à *p3*. Esta preferência conflita (nó C2) com o conflito C1 e com a proposição *p3*. Dois vértices de informação só podem ser unidos indiretamente via nós I, C ou P.

A condição de aceitabilidade determina se um grafo que representa requisitos elicitados e os argumentos associados são aceitáveis para os participantes na construção do grafo, ou se o grafo ainda precisa de uma discussão mais profunda. Basicamente, cada vértice que recebe uma aresta de um vértice de conflito está sendo atacado. Esse ataque pode ser dominado por regras de preferência.

A Figura 2.7 mostra ainda como aplicar o algoritmo. No primeiro passo, os vértices não atacados são marcados como "A" (aceitáveis). Todos os vértices não marcados que restaram estão sendo potencialmente atacados. Durante o segundo passo, as regras de preferência dominam os vértices atacados, marcando-

os como “AD” (atacados e dominados). Durante o terceiro passo, vértices atacados e não-dominados são marcados como “R” (rejeitados). Depois de aplicar o algoritmo, se qualquer proposição originada no artefato de ER foi rejeitada, o artefato de ER não é válido em relação aos interessados que participaram da discussão.

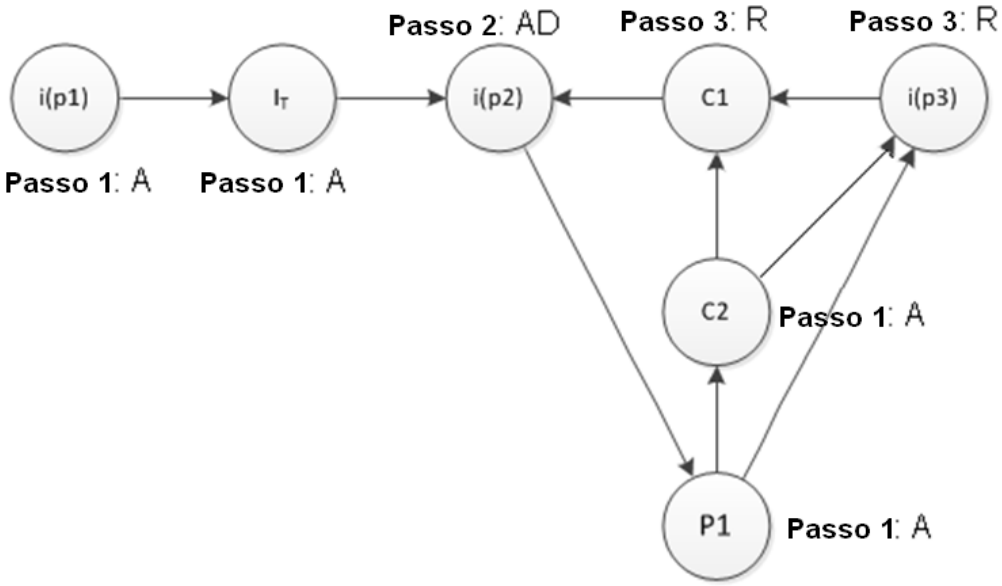

Figura 2.7 - Um exemplo da notação gráfica da linguagem ACE

2.5. Considerações Finais

Apresentamos uma visão geral dos principais conceitos e tecnologias usados como suporte na nossa abordagem, dentre eles destacam-se: os modelos intencionais propostos pelo *framework* i* e pelo NFR Framework; o conceito de transparência bem como o Catálogo de Transparência construído com base na notação de *Softgoal Interdependency Graph* do NFR Framework; os SMAs intencionais; e o *framework* ACE.

O modelo intencional do *framework* i* é largamente utilizado em nossa abordagem, uma vez que os requisitos e os requisitos de transparência do software são modelados de forma intencional através desse *framework*.

O modelo intencional do NFR Framework é utilizado pelo Catálogo de Transparência de Software. Nossa abordagem entrelaça SIGs do NFR Framework aos requisitos do software para introduzir os conceitos relacionados à transparência de software

Nossa abordagem visa operacionalizar a Transparência de Software, de modo a produzir um software transparente. Para isso, utilizamos os conceitos relacionados à Transparência de Software, armazenados no Catálogo de Transparência de Software, para apoiar discussões entre os interessados.

Uma vez que os requisitos e os requisitos de transparência do software são modelados através de *frameworks* intencionais, a implementação do software como um sistema multi-agentes intencional evita o uso de diferentes paradigmas e uma diferença elevada do nível de abstração dos requisitos ao código.

Nossa abordagem incentiva discussões sobre a transparência do software entre os interessados. Os argumentos trocados nessa discussão são modelados e analisados através do *framework* ACE. Algoritmos desse *framework* permitem que se determine, formalmente, a existência de um consenso entre os interessados.