7.
Abordagem Integrada para o Desenvolvimento de Software Transparente

Esse capítulo descreve a abordagem que integra nossos esforços, apresentados nos Capítulos 3 a 6, para o desenvolvimento de software transparente. Para isso, modelamos nossa abordagem através de um SADT que representa as atividades propostas. Apresentamos também a aplicação dessa abordagem em um estudo de caso, o Lattes-Scholar.

Os Capítulos 3 a 6 apresentam os nossos esforços relacionados aos quatro desafios encontrados quando tentamos construir um software transparente. Cada um desses capítulos descreve, de forma independente, os trabalhos relacionados a apenas um dos desafios. Os capítulos estão ordenados segundo a ordem cronológica dos trabalhos.

Nossos trabalhos seguiram uma ordem aparentemente inversa da ordem que seria esperada. A ordem que aparentaria ser a mais natural, ou *top-down*, seria primeiro elicitar e validar requisitos de transparência; depois, tratar questões relacionadas à pré-rastreabilidade desses requisitos e, por último, definir um conjunto de heurísticas para guiar o desenvolvimento do SMA intencional. A aparente inversão se deve ao fato de termos abordado a transparência de software enquanto desenvolvíamos o software dos estudos de caso. Portanto, nossos trabalhos foram realizados de uma forma *bottom-up,* ao longo de uma série de experimentos. Os primeiros trabalhos foram concentrados em anexar os requisitos ao código, através de heurísticas transformacionais. Logo nos deparamos com o segundo grande desafio, conseguir que os agentes raciocinassem em termos de metas flexíveis. Uma vez que as primeiras versões das heurísticas e da máquina de raciocínio qualitativa estavam disponíveis, tratamos do uso do Catálogo de Transparência de Software em um processo de argumentação para elicitar e validar relativamente requisitos de transparência. Por último, preocupamos-nos com a pré-rastreabilidade dos requisitos de transparência, uma vez que esses eram a base para a construção do software transparente.

Nesse capítulo, apresentamos uma abordagem integrada para o desenvolvimento intencional de software transparente. A abordagem é modelada através de um SADT que representa as atividades propostas. Cada atividade aplica dois ou mais trabalhos relacionados aos desafios para a transparência de software. Duas atividades formam um ciclo que permite a retro-alimentação, definindo, assim, um processo iterativo e incremental de desenvolvimento.

O principal objetivo é oferecer uma abordagem que integre os nossos esforços de uma forma lógica e clara, baseada em atividades que possam ser introduzidas em processos de desenvolvimentos de software já existentes.

Aplicamos a abordagem integrada em um estudo de caso, o Lattes-Scholar. Mostramos como um dos requisitos de transparência mais importantes, relacionado à disponibilidade do software como uma aplicação *Web*, determinou a arquitetura a ser utilizada, uma arquitetura híbrida de páginas *Web* com SMA intencional. Além disso, apresentamos alguns dos artefatos produzidos em cada atividade principal.

Esse capítulo está dividido em seções. A Seção 7.1 mostra a abordagem que integra os trabalhos apresentados nos Capítulos 3 a 6. A Seção 7.2 descreve o estudo de caso Lattes-Scholar e mostra alguns dos artefatos produzidos em cada uma das atividades. Finalmente, a Seção 7.3 apresenta as considerações finais do capítulo.

7.1. Abordagem Proposta para Desenvolvimento Intencional Baseada em Argumentação

A Figura 7.1 ilustra a abordagem proposta para desenvolvimento intencional baseada em argumentação e modelada em SADT (MARCA e McGOWAN 1987). Três atividades principais – Argumentar, Desenhar e Implementar – são realizadas sucessivamente em um processo iterativo incremental. Os artefatos produzidos em uma iteração são entradas para as atividades da iteração posterior, através das atividades "Armazenar" e "Consultar", resultando em uma retro-alimentação que representa o refinamento dos artefatos.

Cada uma das atividades da Figura 7.1 aplica técnicas, métodos e/ou modelos apresentados anteriormente nos Capítulos 3 a 6 para lidar com os

desafios em relação a transparência de software. A atividade Argumentar aplica argumentação e validação relativa (Capítulo 5) para elicitar, modelar e validar os requisitos de transparência; e modelos de pré-rastreabilidade ITrace (Capítulo 6) para rastrear os artefatos da Engenharia de Requisitos de volta às reuniões e aos argumentos dos interessados. A atividade Desenhar aplica heurísticas transformacionais de desenho (Capítulo 3), desenha metas flexíveis como variáveis nebulosas (Capítulo 4), e rastreia as atividades de desenho com modelos de pré-rastreabilidade ITrace (Capítulo 6). A atividade Implementar aplica heurísticas transformacionais de implementação (Capítulo 3) e a máquina de raciocínio qualitativa baseada em lógica nebulosa (Capítulo 4) para implementar agentes de software colaborativos. As atividades de implementação são rastreadas através de modelos ITrace (Capítulo 6). As atividades Armazenar e Consultar manipulam modelos intencionais, grafos de argumentação (Capítulo 5), especificações BDI (Capítulos 3 e 4), cenários (Capítulos 3 e 4), código anotado de SMAs (Capítulo 4), e modelos ITrace (Capítulo 6).

A atividade Argumentar baseia-se em uma série de discussões sobre a transparência da aplicação, como exposto no Capítulo 5. Essas discussões entre os interessados são mediadas pelo engenheiro de requisitos. Os requisitos são representados em um modelo intencional, no caso, modelos de Dependência Estratégica (*Strategic Dependency* - SD) e de Rationale Estratégico (*Strategic Rationale* - SR) do *framework* i* (vide Capítulo 2 – Seção 2.1). Cada iteração visa discutir os requisitos em relação a uma meta flexível que contribui para a transparência, extraída do CTS. Os requisitos e a meta flexível relacionada à transparência são expressos como proposições na linguagem ACE e são atacados ou apoiados por argumentos colocados pelos interessados, resultando em um grafo. Esse grafo é mantido na ferramenta C&L (C&L 2011). Algoritmos do *framework* ACE analisam se a discussão chegou ao fim em um determinado momento ou se uma discussão mais profunda ainda é necessária. Se os interessados chegaram a um consenso, as proposições da linguagem ACE são traduzidas de volta para o *framework* i* e uma nova versão dos requisitos é obtida. Essa nova versão tem a sua transparência relativamente validada, ou seja, válida sob os pontos de vista dos interessados que participaram da discussão. Cada interação social entre os interessados é rastreada através de um modelo ITrace. Todos os artefatos gerados são inseridos em um Wiki.

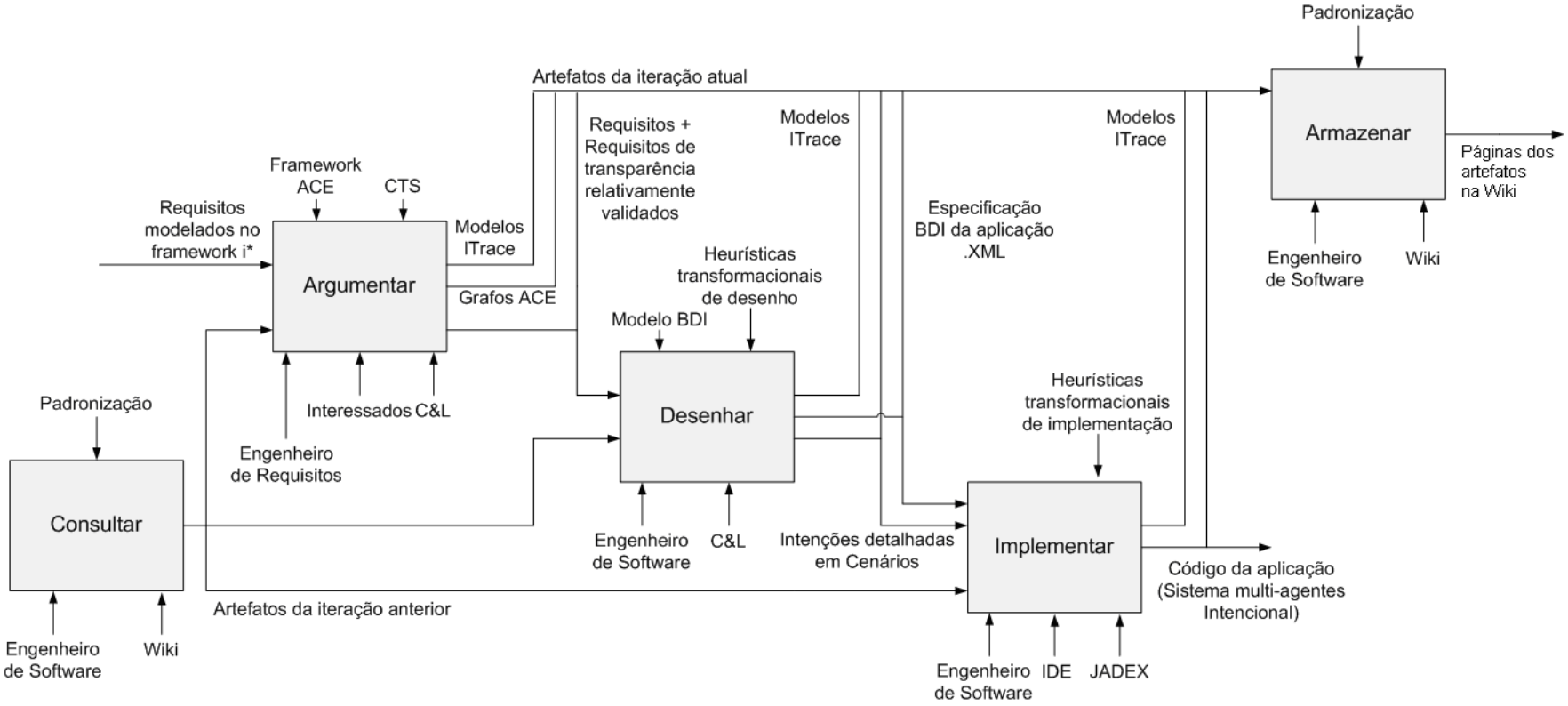

Figura 7.1 - O processo de desenvolvimento proposto modelado em SADT

A atividade Desenhar compreende o desenho da aplicação, baixando o nível de abstração através de heurísticas que transformam os requisitos intencionais em uma especificação da aplicação no modelo BDI, como exposto no Capítulo 3 – Seção 3.2. Essa especificação possui lacunas que são preenchidas manualmente pelo engenheiro de software ou observando se as mesmas já haviam sido preenchidas na especificação da iteração anterior. As intenções são descritas com mais detalhes através de cenários e sub-cenários, mantidos na ferramenta C&L. Todos os artefatos gerados nessa atividade também são inseridos no Wiki. As heurísticas utilizadas na atividade Desenhar são discutidas com mais profundidade no Capítulo 3 – Seção 3.2.

A atividade Implementar aplica heurísticas para transformar a especificação da aplicação no modelo BDI e os cenários, que descrevem as intenções dos agentes, em código de sistemas multi-agentes para o *framework* JADEX. Esse código é uma biblioteca Java que contém os ADFs e os planos dos agentes obtidos. Os episódios dos cenários são inseridos como comentários no código dos planos, orientando quais ações devem ser programadas e onde inserir esses trechos de código Java. Trechos já implementados em iterações anteriores podem ser reaproveitados parcial ou integralmente. O trabalho dos engenheiros de software é acompanhado através de modelos ITrace e todos os artefatos gerados nessa atividade são inseridos no Wiki. As heurísticas utilizadas na atividade Implementar são discutidas com mais profundidade no Capítulo 3 – Seção 3.2.

Os artefatos produzidos pelas três atividades principais, Argumentar, Desenhar e Implementar, são inseridos em um Wiki, padronizado e aberto a todos os participantes do processo. Os artefatos são subdivididos por iteração e por atividade, sendo expostos no Wiki seguindo a ordem cronológica das interações sociais que os justificam. O Wiki também mantém elos para a ferramenta C&L de modo a facilitar o acesso aos grafos ACE e aos cenários mantidos nessa ferramenta.

7.2. Estudo de Caso: Aplicação *Web* Lattes-Scholar

Utilizamos nossa abordagem integrada para o desenvolvimento intencional de software transparente em um estudo de caso, o Lattes-Scholar (Lattes-Scholar

2011a; Lattes-Scholar 2011b). O Lattes-Scholar é uma aplicação Web para ordenar listas de publicações científicas pelo número de citações. As listas de publicações científicas são obtidas a partir de currículos de pesquisadores mantidos no Lattes (Lattes 2011), uma base de dados de currículos e instituições de pesquisa e tecnologia mantida pelo Governo Brasileiro. O número de citações é fornecido pelo Google Scholar, um serviço muito conhecido pela comunidade de pesquisa científica. O Lattes-Scholar compõe esses dois serviços, Lattes e Google Scholar, para ordenar pelo número de citações as publicações listadas no currículo de um pesquisador. Dessa forma, o público alvo desse software é toda a comunidade de pesquisadores científicos brasileiros. O Apêndice A apresenta exemplos de uso através de telas do Lattes-Scholar.

O Lattes-Scholar originou-se de um serviço implementado em páginas *Web,* na linguagem PHP, para o *Workshop* de Engenharia de Requisitos (WER), o WER-Papers (WERPAPERS 2011). Esse serviço utiliza o Google Scholar para apresentar o número de citações de cada artigo já publicado pelo WER. Posteriormente, esse serviço foi estendido com a consulta ao Lattes, e implementado na linguagem Lua. A versão mais recente, anterior à nossa, implementou o Lattes-Scholar como um SMA comportamental em nuvem. Contudo, o foco de todas essas versões era prover as funcionalidades definidas em tempo de desenho, não necessariamente de forma transparente.

As seções seguintes, Seções 7.2.1 a 7.2.4, descrevem o desenvolvimento do Lattes-Scholar seguindo a nossa abordagem, apresentando alguns dos artefatos produzidos ao longo de cada atividade.

7.2.1. Atividade Argumentar

A atividade Argumentar foi realizada em uma série de reuniões que promoveram discussões entre o grupo de interessados no Lattes-Scholar. Essa série de reuniões concentrou a atenção do Grupo de Transparência de Software ao longo de dois meses, em um total de seis reuniões para a discussão de metas flexíveis relacionadas à transparência, além de uma reunião para a apresentação do modelo de pré-rastreabilidade ITrace e um tutorial sobre pré-rastreabilidade.

A transparência de software possui 28 metas flexíveis relacionadas. Essas 28 metas flexíveis dividem-se em cinco grandes grupos, de acordo com o SIG de

Transparência: as metas flexíveis relacionadas à Acessibilidade, à Usabilidade, ao Entendimento, à Informatividade ou à Auditabilidade. O estudo de caso foi elaborado de forma que (i) todas as metas flexíveis relacionadas à Acessibilidade fossem discutidas, uma vez que a acessibilidade é uma meta flexível crítica para a transparência de software; e (ii) pelo menos uma meta flexível pertencente a cada um dos cinco grandes grupos fosse discutida, permitindo discussões abrangentes e diferenciadas.

A primeira reunião discutiu a principal meta flexível para a Acessibilidade – a Disponibilidade. Alguns argumentos também abordaram a Portabilidade e a Publicidade. Um período considerável da discussão foi focado em esclarecer as diferenças entre Disponibilidade e Operabilidade, o que gerou uma evolução em dois padrões de requisitos do CTS. Os padrões de requisitos "Objetivos de Transparência de Software", "Questões de Disponibilidade" e "Questões de Operabilidade" foram recuperados do CTS e apoiaram a discussão dos interessados sobre a Disponibilidade.

A segunda reunião concentrou-se na discussão sobre a Portabilidade das informações e a Publicidade do Lattes-Scholar, de forma a concluir a discussão sobre Acessibilidade. A Figura 7.2 mostra um *storyboard* produzido com quadros dos vídeos da segunda reunião. Os padrões de requisitos "Objetivos de Transparência de Software", "Questões de Portabilidade" e "Questões de Publicidade" foram recuperados do CTS e apoiaram a discussão dos interessados.

Figura 7.2 - *Storyboard* produzido com quadros dos vídeos da segunda reunião

A terceira reunião discutiu a Rastreabilidade e, por conseqüência, a Pré-Rastreabilidade dos artefatos produzidos no desenvolvimento do Lattes-Scholar. A Rastreabilidade é uma meta flexível relacionada à Auditabilidade. Os padrões de requisitos "Objetivos de Transparência de Software" e "Questões de Rastreabilidade" foram recuperados do CTS e apoiaram a discussão dos interessados sobre Rastreabilidade.

A quarta reunião apresentou aos interessados protótipos de interfaces para que fosse possível a discussão sobre a Amigabilidade das páginas *Web* do Lattes-Scholar. A Amigabilidade é uma meta flexível relacionada à Usabilidade. Os padrões de requisitos "Objetivos de Transparência de Software" e "Questões de Amigabilidade" foram recuperados do CTS e apoiaram a discussão dos interessados sobre Amigabilidade.

A quinta reunião concentrou-se na meta flexível Atualização. A Atualização está relacionada à meta flexível Informatividade. Os padrões de requisitos "Objetivos de Transparência de Software" e "Questões de Atualização" foram recuperados do CTS e apoiaram a discussão dos interessados sobre Atualização.

A sexta reunião discutiu os requisitos e o modelo arquitetural do Lattes-Scholar em relação à meta flexível Concisão. A Concisão está relacionada à meta flexível Entendimento. Os padrões de requisitos "Objetivos de Transparência de Software" e "Questões de Atualização" foram recuperados do CTS e apoiaram a discussão dos interessados sobre Atualização.

O autor da abordagem (engenheiro de requisitos) participou das reuniões no papel de mediador, permitindo a livre discussão entre os outros interessados. As principais funções do mediador foram: apresentar aos interessados os padrões de requisitos obtidos a partir do Catálogo de Transparência de Software e os artefatos do Lattes-Scholar; e anotar em um quadro branco os principais argumentos dos interessados para que os outros interessados pudessem referenciá-los, argumentando contra ou a favor.

Todas as reuniões foram gravadas em vídeo. As gravações permitiram a pré-rastreabilidade dos grafos de argumentação da linguagem ACE. Todos os nós que representam argumentos dos interessados receberam etiquetas que referenciam os vídeos e associam o argumento ao interessado, como pode ser observado na Figura 7.3. Essa figura apresenta um grafo de argumentos na linguagem ACE que representa a discussão ocorrida na primeira reunião sobre a disponibilização do

Lattes-Scholar como uma aplicação *Web*, um dos principais requisitos de transparência. Esse requisito determinou a arquitetura do Lattes-Scholar, uma vez que seriam necessárias páginas *Web* integradas com um SMA intencional.

Modelos ITrace também fazem a pré-rastreabilidade dos grafos de argumentação, rastreando-os de volta à reunião que representam, aos interessados que participaram da discussão, às fontes de informação e aos recursos utilizados. A Figura 7.4 mostra o modelo ITrace para a primeira reunião. Esse modelo oferece pré-rastreabilidade para três grafos ACE, entre eles o grafo da Figura 7.3.

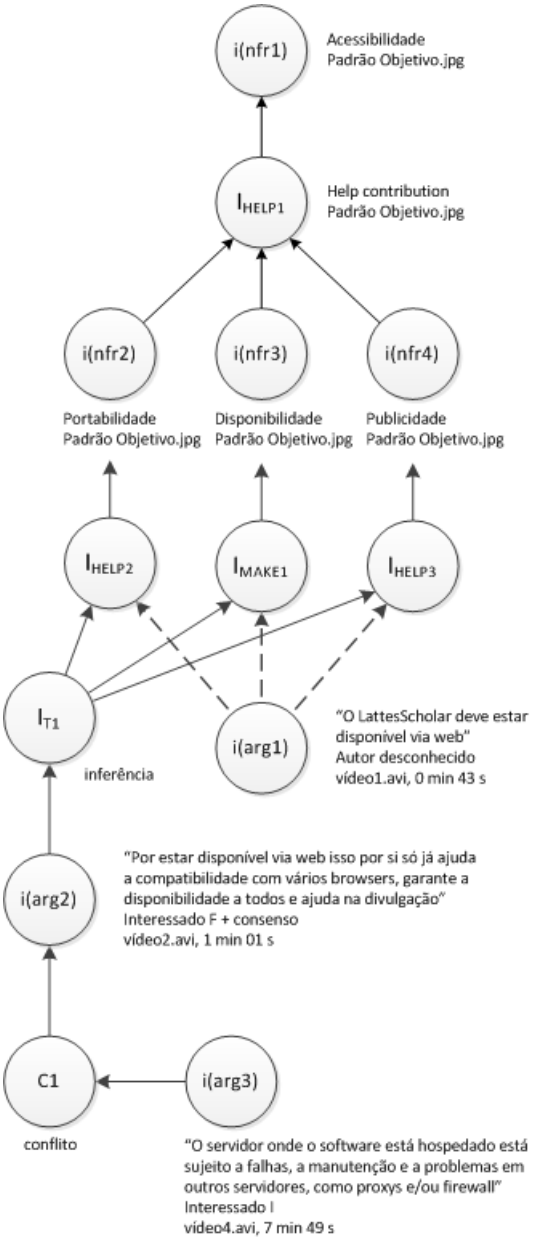

Figura 7.3 - O grafo ACE que representa a discussão ocorrida na primeira reunião sobre a disponibilização do Lattes-Scholar como uma aplicação *Web*

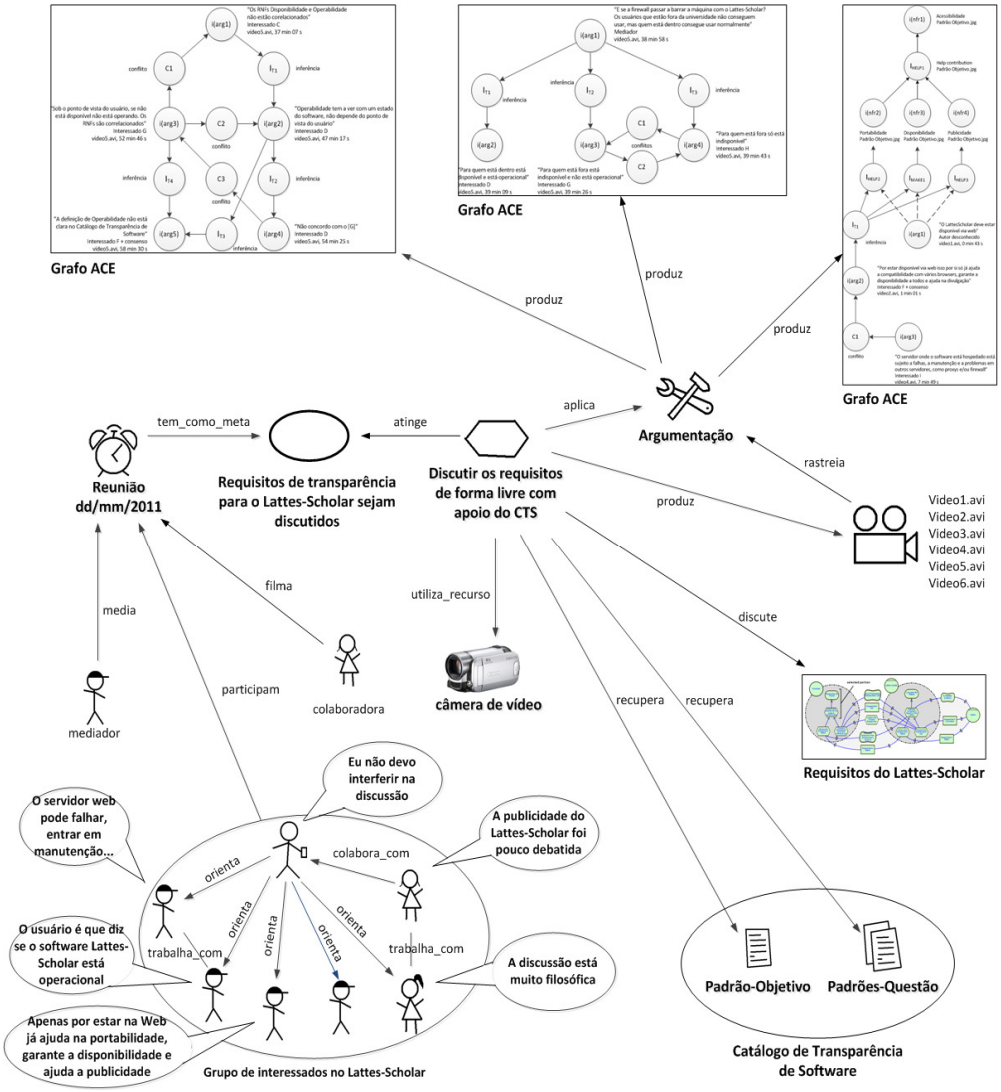

Figura 7.4 - Modelo ITrace da primeira reunião

Com base nas discussões da primeira e da segunda reuniões sobre a acessibilidade do Lattes-Scholar como uma aplicação *Web*, produziu-se, por exemplo, um modelo de atores estratégicos (*Strategic Actors* - SA). Esse modelo apresenta os atores envolvidos no ambiente do software e as posições e papéis que assumem. Um pesquisador, por exemplo, assume a posição de autor de publicações científicas. Como os pesquisadores são cidadãos, eles também podem assumir a posição de consumidores das informações fornecidas pelo Lattes-Scholar. O modelo SA teve sua rastreabilidade discutida na terceira reunião e sua concisão discutida na sexta reunião. A versão final do modelo SA é apresentada na Figura 7.5.

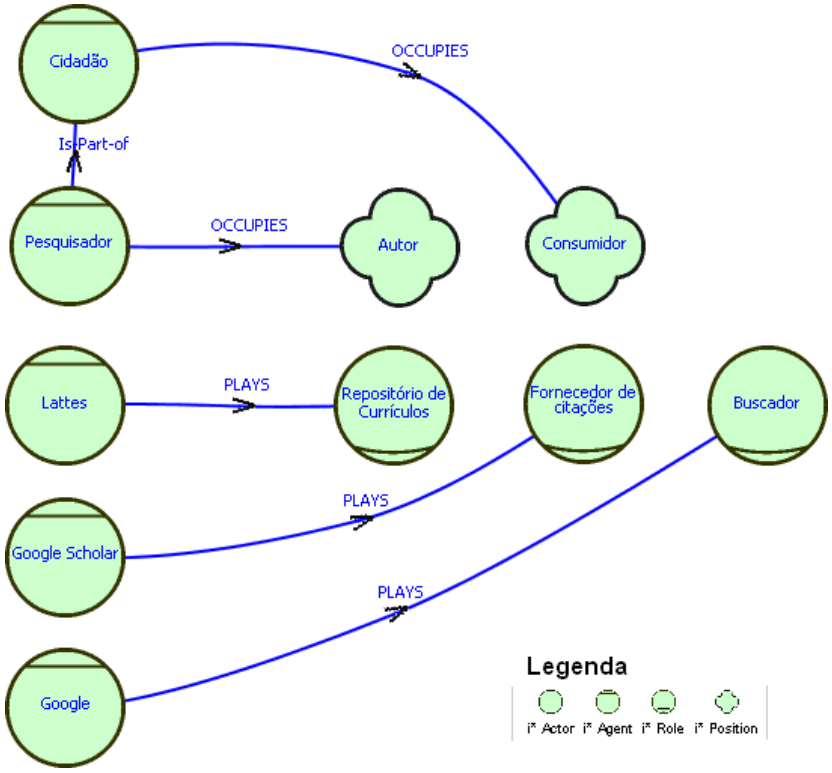

Figura 7.5 - Modelo de Atores Estratégicos do Lattes-Scholar

O modelo SA do Lattes-Scholar, apresentado na Figura 7.5, foi a base para a construção do modelo SD Situation (Oliveira 2008) para a situação "Recuperar Citações", cuja versão final é apresentada na Figura 7.6. Quatro elos de rastreabilidade, ou rastros, ligam esses dois modelos: (i) modelo SA – modelo SD, obtido a partir da padronização dos nomes [nome do modelo]-SA-[versão] e [nome do modelo]-SD-[versão], onde [nome do modelo] é "Recuperar Citações"; (ii) posição "Consumidor" do modelo SA - posição de mesmo nome no modelo SD; (iii) papel "Repositório de Currículos" do modelo SA - papel de mesmo nome no modelo SD; e (iv) papel "Fornecedor de Citações" do modelo SA - papel de mesmo nome no modelo SD. Esse modelo já considera a disponibilidade em uma página *Web* e mostra as dependências entre os atores estratégicos quando, a partir do currículo do pesquisador, o "SMA do Lattes-Scholar" obtém os nomes das seções que contém publicações científicas. Esses nomes são apresentados pela "Página Web do Lattes-Scholar" ao "Consumidor", que pode então selecionar as seções que deseja. O "SMA do Lattes-Scholar" recupera as publicações das seções selecionadas e solicita o número de citações de cada uma das publicações ao "Fornecedor de Citações".

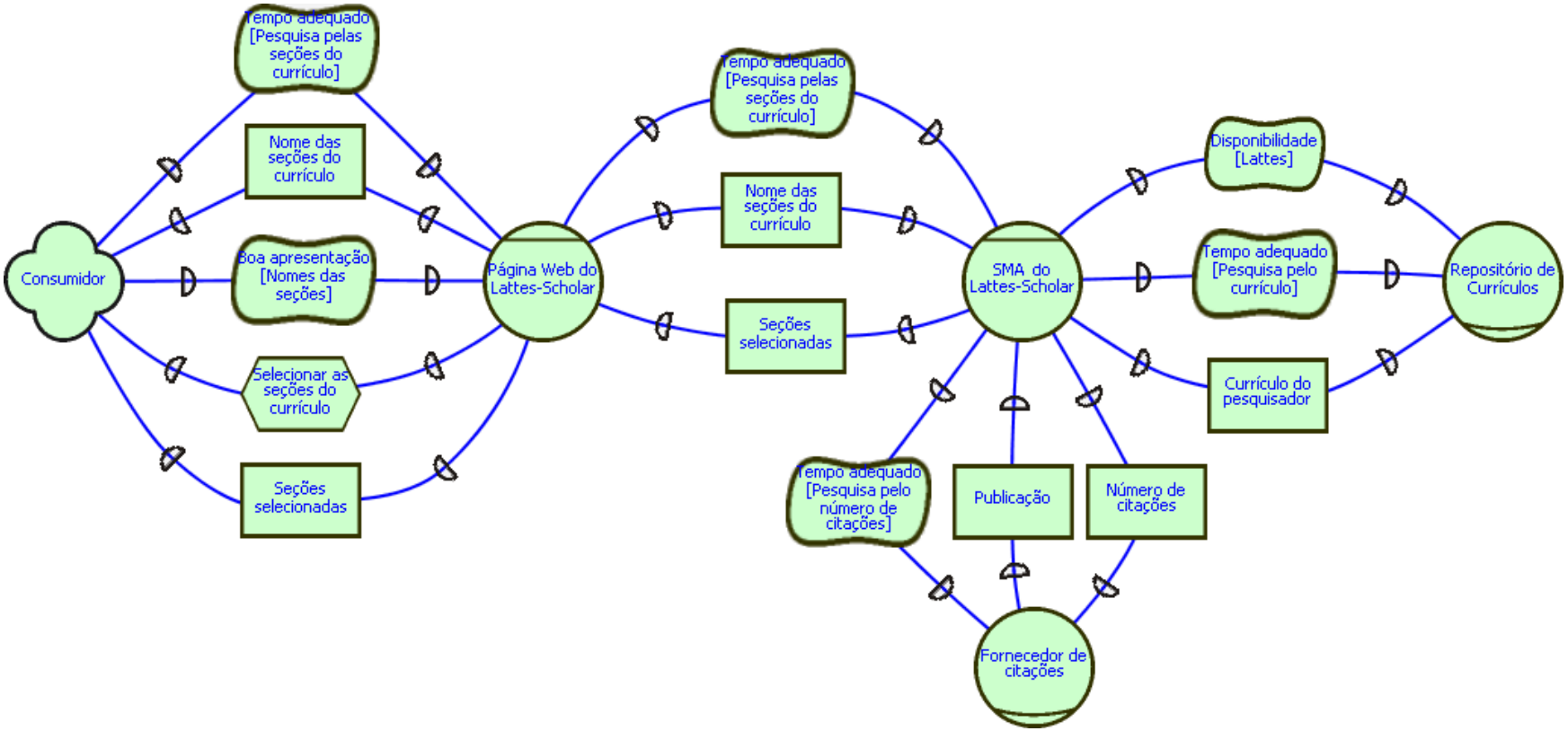

Figura 7.6 - Modelo de Dependências Estratégicas do Lattes-Scholar para a situação “Recuperar citações”

Os modelos SA e SD das Figuras 7.5 e 7.6, respectivamente, foram a base para o modelo arquitetural intencional do Lattes-Scholar, desenhado na atividade Desenhar.

7.2.2. Atividade Desenhar

A atividade Desenhar compreende o trabalho do engenheiro de software de aplicar as heurísticas transformacionais de desenho, expostas no Capítulo 3 – Seção 3.2, nos requisitos (incluindo os requisitos de transparência) do software elicitados ou evoluídos na atividade Argumentar. Além disso, a atividade Desenhar inclui o trabalho de especificar, através de cenários, os episódios (ou *script* de ações) de cada uma das tarefas incluídas nos modelos i*.

As especificações BDI produzidas pelas heurísticas transformacionais de desenho possuem algumas lacunas, como explicado no Capítulo 3 – Seção 3.2. O trabalho do engenheiro de software na atividade Desenhar também inclui preencher manualmente essas lacunas ou recuperar, através da atividade Consultar, o trecho já preenchido em iterações anteriores.

A Figura 7.7 apresenta o modelo arquitetural em i* do SMA do Lattes-Scholar. Esse modelo arquitetural foi desenhado com base nos modelos SA e SD construídos na atividade Argumentar, Figuras 7.5 e 7.6, respectivamente. Nesse modelo, o agente "Gerente do SMA do Lattes-Scholar" coordena o trabalho de três agentes: o que assume o papel "Repositório de Currículos" e faz a interface com o Lattes; o agente "Extrator de Seções", responsável por manipular o currículo do pesquisador, extraindo as seções selecionadas e listando as publicações de cada seção; e o agente que assume o papel "Fornecedor de Citações", que solicita ao Google Scholar, uma a uma, as citações de cada publicação.

Ao todo, cinco rastros ligam o modelo arquitetural ao modelo SA da Figura 7.5: (i) modelo SA – modelo arquitetural, obtido a partir da padronização de nomes [nome do modelo]-SA-[versão] e [nome do modelo]-Arq-[versão], onde o nome do modelo é "Recuperar Citações"; (ii) papel "Fornecedor de Citações" do modelo SA – papel no modelo arquitetural; (iii) papel "Repositório de Currículos" do modelo SA – papel no modelo arquitetural; (iv) agente "Lattes" do modelo SA – agente no modelo arquitetural; e (v) agente "Google Scholar" do modelo SA –

agente no modelo arquitetural. Além disso, 14 rastros ligam o modelo arquitetural ao modelo SD da Figura 7.6. O principal rastro é obtido a partir da padronização de nomes [nome do modelo]-SD-[versão] e [nome do modelo]-Arq-[versão]. Os outros rastros são oriundos da padronização dos nomes dos agentes, papéis, metas flexíveis, metas e recursos. Basicamente, todas as abstrações da Figura 7.6 estão presentes no modelo arquitetural, com exceção da posição "Consumidor" (externa ao software) e das respectivas dependências estratégicas.

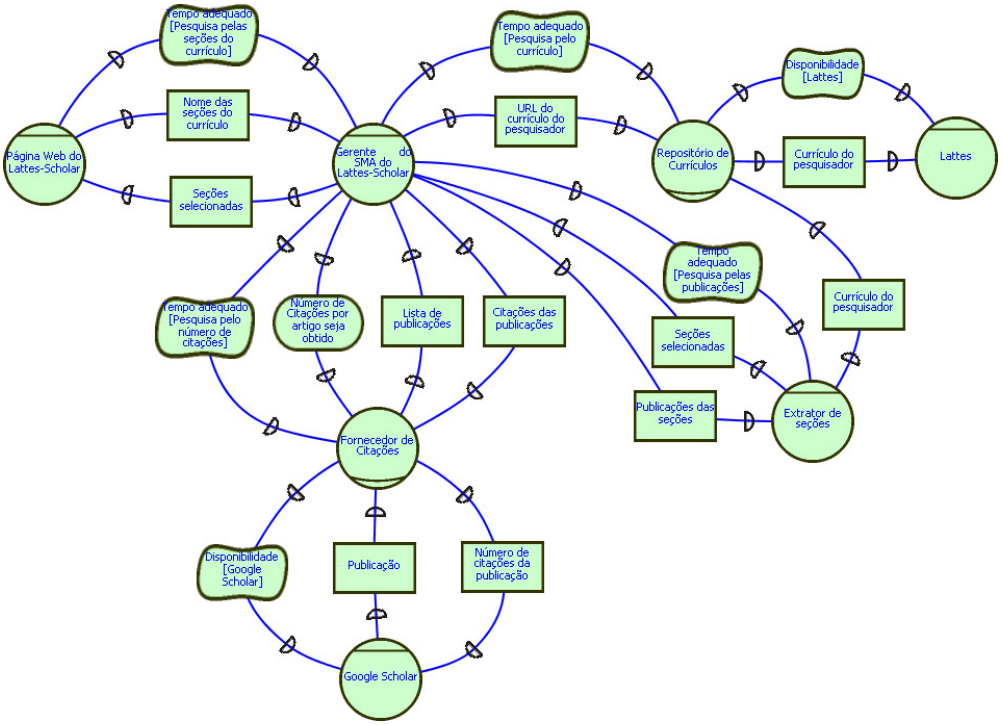

Figura 7.7 - Modelo arquitetural em i* do SMA do Lattes-Scholar para a situação "Recuperar Citações"

A Figura 7.8 mostra mais detalhes do papel "Fornecedor de Citações" através de um modelo de *Rationale* Estratégico (*Strategic Rationale* - SR) parcial. O modelo SR apresenta duas alternativas para a meta "Número de citações por publicação seja conhecido": "Requisitar o número de citações para o Google Scholar" ou "Delegar a meta a um agente móvel". Cada alternativa envolve impactos diferentes sobre as metas flexíveis "Performance" e "Evitar muitas requisições [Google Scholar]". A escolha entre essas alternativas em tempo de execução já foi discutida no Capítulo 4. Sete rastros ligam o modelo SR ao modelo arquitetural da Figura 7.7. Esses rastros referem-se ao agente "Google Scholar", ao papel "Fornecedor de Citações", e às dependências estratégicas entre eles.

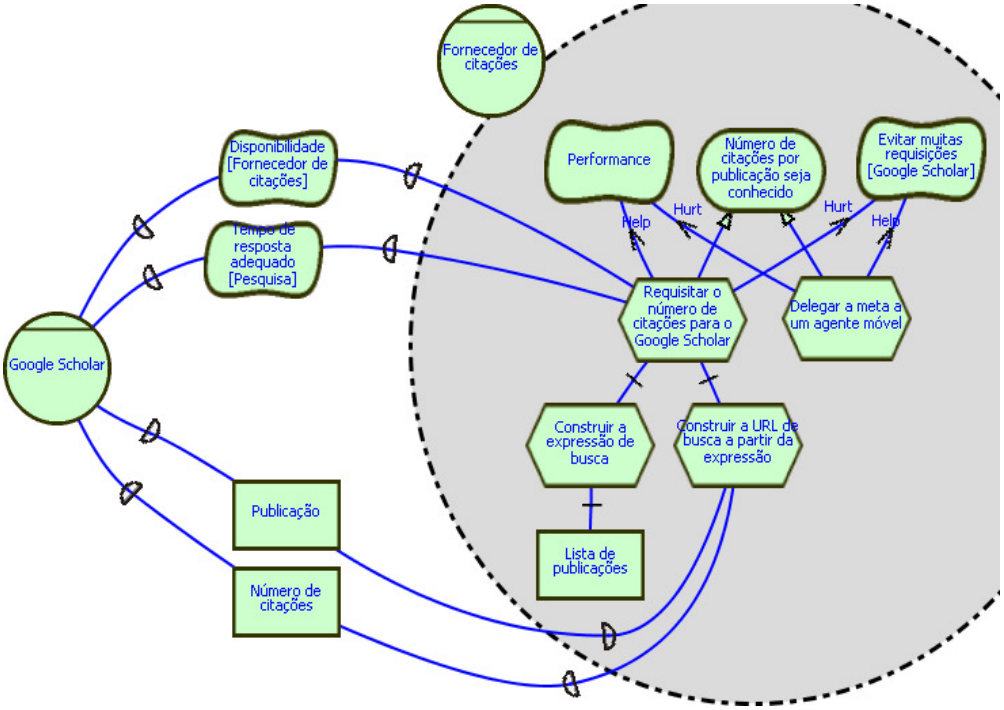

Figura 7.8 - Modelo de *Rationale* Estratégico parcial do papel "Fornecedor de Citações" na situação "Recuperar Citações"

A tarefa "Requisitar o número de citações para o Google Scholar" do papel "Fornecedor de Citações", Figura 7.8, foi detalhada através de um *script* de ações. Esse *script* foi especificado nos episódios do cenário (Leite et al. 1997), apresentado na Figura 7.9. Assim como a tarefa "Requisitar o número de citações para o Google Scholar" é decomposta em duas tarefas, "Construir a expressão de busca" e "Construir a URL de busca a partir da expressão", o cenário "Requisitar o número de citações para o Google Scholar" utiliza dois sub-cenários, nomeados de acordo com as sub-tarefas que representam. O cenário foi criado pela heurística transformacional de desenho número 07 (vide Capítulo 3 – Seção 3.2) e, portanto, existe um rastro entre a tarefa do modelo SR da Figura 7.8 e o cenário da Figura 7.9, do tipo tarefa-cenário. Esse rastro define o nome do arquivo do cenário como "Recuperar Citações\Fornecedor de Citações\Requisitar o número de citações para o Google Scholar-v0.1.cen". Outros dois rastros ligam a tarefa do modelo SR ao conteúdo da *tag* "script" na especificação BDI e o cenário à mesma *tag*.

```
Título: Requisitar o número de citações para o Google Scholar
Atores: Papel Fornecedor de citações, Agente Google Scholar
Sub-cenários: Construir a expressão de busca,
              Construir a URL de busca a partir da expressão
Meta: Requisitar o número de citações para o Google Scholar
Metas flexíveis impactadas: Performance,
          Evitar muitas requisições [Google Scholar]
Recursos: lista de publicações, publicação, número de citações,
          expressão de busca, URL de busca.
Episódios:
1-Para cada publicação da lista de publicações:
2-Construir a expressão de busca
3-Construir a URL de busca a partir da expressão
4-Fazer uma requisição HTTP ao Google Scholar usando a URL
  de busca
5-Recuperar o código fonte (HTML) da página Web de resposta
6-Filtrar o número de citações do código em HTML
7-Somar o número de citações caso encontre duas ou mais
  instâncias da mesma publicação na página Web de resposta
```

Figura 7.9 - A tarefa “Requisitar o número de citações para o Google Scholar” detalhada como um cenário

Aplicando as heurísticas transformacionais de desenho, o Modelo de *Rationale* Estratégico do papel “Fornecedor de Citações”, Figura 7.8, é traduzido para uma especificação BDI de um agente não executável, Figura 7.10. O principal rastro é do tipo modelo SR-especificação BDI, obtido através da padronização de nomes [nome do modelo]-SR-[versão] e [nome do modelo]-BDI-[versão]. Algumas lacunas, como o *script* das intenções, são preenchidas com cenários, como o da Figura 7.9. Outras, como o tipo do desejo, são preenchidas manualmente pelo engenheiro de software. Alguns recursos, como “Expressão de busca”, são implícitos no modelo de *Rationale* Estratégico e devem ser criados pelo engenheiro de software como crenças na especificação BDI.

Outros onze rastros ligam a especificação BDI ao modelo SR da Figura 7.8: um par de rastros relacionado à *tag* “agent”; sete rastros relacionados às crenças; um rastro relacionado à meta; e outro rastro relacionado à intenção. Além disso, outro rastro liga o conteúdo da *tag* “script” ao respectivo cenário.

As especificações BDI dos atores, agentes, papéis e posições, detalhados em modelos SR do *framework* i*, são as principais entradas para a atividade seguinte, a atividade Implementar. O detalhamento das tarefas/intenções em cenários também são entradas para essa atividade.

```
<bdimodel name="lattesscholar"
  package="lattesscholar.sma.fornecedordecitacoes">
  <agents>
    <agent name="Fornecedor de Citações" runnable="no">
      <beliefs>
        <belief name="Expressão de busca" type="String"/>
        <belief name="URL de busca" type="String"/>
        <beliefset name="Lista de publicações" type="Publicacoes"/>
        <beliefset name="Lista de Número de citações" type="Integer"/>
        <beliefset name="tasks" type="Task"/>
        <beliefset name="contributions" type="Contribution"/>
        <beliefset name="softgoals" type="Softgoal"/>
      </beliefs>
      <desires>
        <desire name="Número de citações por publicação seja conhecido"
          type="achieve"/>
      </desires>
      <intentions>
        <intention name="Requisitar o número de citações para o Google
          Scholar">
          <desire>Número de citações por publicação seja conhecido
          </desire>
          <script lang="scenarios">
            ...
          </script>
        </intention>
        <intention name="Delegar a meta a um agente móvel">
          <desire>Número de citações por publicação seja conhecido</desire>
          <script lang="scenarios">
            ...
          </script>
        </intention>
      </intentions>
    </agent>
  </agents>
</bdimodel>
```

Figura 7.10 - A especificação BDI do papel “Fornecedor de Citações” como um agente não executável

7.2.3. Atividade Implementar

A atividade Implementar compreende o trabalho do engenheiro de software de, em um primeiro momento, aplicar as heurísticas transformacionais de implementação, expostas no Capítulo 3 – Seção 3.2, nas especificações BDI dos agentes intencionais, produzidas na atividade Desenhar. As especificações BDI já incluem os cenários e sub-cenários que detalham as tarefas/intenções dos agentes. Em um segundo momento, o engenheiro de software deve implementar o software a partir dos *templates* dos planos e dos ADFs dos agentes. Expondo de uma forma mais simples, a aplicação das heurísticas produz uma “casca” vazia dos agentes, mas que já contém todos os arquivos XML, as classes, e os protótipos dos métodos necessários à implementação. Cabe ao engenheiro de software preencher trechos desses arquivos ou definir o corpo desses métodos com o código do SMA.

A Figura 7.11 mostra o cabeçalho, os *imports* e as capacidades definidas no ADF "FornecedorDeCitacoes.capability.xml" produzido pelas heurísticas transformacionais de implementação a partir da especificação BDI do agente não executável "Fornecedor de Citações" (Figura 7.10). O rastro entre a especificação BDI e a capacidade JADEX é obtido através da heurística transformacional de implementação 02 (vide Capítulo 3 – Tabela 3.2), que define o par (agente não executável BDI, capacidade JADEX) e a padronização de nomes "[nome do agente não executável]" e "[nome do agente não executável].capability.xml". Na Figura 7.11, o último *import*, "istar.metamodel.MetaPlan", referencia o meta-plano para agentes intencionais já implementado. Esse meta-plano contém todo o código necessário para executar a máquina de raciocínio qualitativa baseada em lógica nebulosa, Capítulo 4 – Seção 4.2. Esse meta-plano assume ainda que o agente possui a capacidade "ReasoningEngine", referenciada na *tag* "capabilities".

```
<capability xmlns="http://jadex.sourceforge.net/jadex"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                        http://jadex.sourceforge.net/jadex-0.96.xsd"
    name="FornecedorDeCitacoes"
    package="lattescholar.sma.fornecedordecitacoes">

    <imports>
        <import>jadex.util.*</import>
        <import>jadex.adapter.fipa.*</import>
        <import>istar.metamodel.MetaPlan</import>
    </imports>

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

Figura 7.11 - ADF da capacidade "FornecedorDeCitacoes", produzido pelas heurísticas transformacionais de implementação

As crenças da capacidade "FornecedorDeCitacoes" são apresentadas na Figura 7.12. Essas crenças foram produzidas pelas heurísticas transformacionais de implementação e assemelham-se às da especificação BDI, porém, com algumas diferenças significativas, tais como: (i) como são crenças de uma capacidade a ser utilizada por um agente executável, as crenças que são compartilhadas entre a capacidade e o agente possuem a *tag* exported="true", como se fossem parâmetros de entrada ou saída; (ii) os tipos determinados na especificação BDI não são,

necessariamente, compatíveis com os tipos oferecidos pela linguagem Java, utilizada pelo *framework* JADEX e, portanto, devem ser convertidos; e (iii) a linguagem Java e o *framework* JADEX utilizam uma padronização dos nomes que não é, necessariamente, igual às utilizadas nos modelos i*, nas especificações BDI e nos cenários. Assim, o nome "Lista de publicações", por exemplo, é convertido para "lista_de_publicacoes". O rastro entre as crenças da especificação BDI e as crenças da capacidade JADEX seguem essa conversão padronizada de nomes.

A Figura 7.12 também mostra três conjuntos de crenças que são definidos em todos os agentes e capacidades produzidos pela nossa abordagem: "tasks", "contributions" e "softgoals". Esses três conjuntos de crenças são fundamentais para o funcionamento da máquina de raciocínio qualitativa, uma vez que esses conjuntos de crenças contêm todas as informações sobre metas flexíveis que estavam presentes no modelo i*. Sem esses conjuntos de crenças, não teríamos como representar essas informações, uma vez que o *framework* JADEX só possui metas booleanas (dois estados, atingida ou não atingida). O conjunto de crenças "softgoals", por exemplo, possui duas crenças (em tempo de execução): a crença "performance" e a crença "evitar_muitas_requisicoes_google_scholar", ambas representando as respectivas metas flexíveis do modelo SR da Figura 7.8.

```
<beliefs>
    <belief name="expressao_de_busca" class="String"/>
    <belief name="url_de_busca" class="String"/>
    <beliefset name="lista_de_publicacoes"
               class="Publicacoes" exported="true"/>
    <beliefset name="lista_de_numero_de_citacoes"
               class="Integer" exported="true"/>
    <beliefset name="tasks" class="Task"/>
    <beliefset name="contributions" class="Contribution"/>
    <beliefset name="softgoals" class="Softgoal"/>
</beliefs>
```

Figura 7.12 - Crenças definidas na capacidade "FornecedorDeCitacoes"

O desejo "Número de citações por publicação seja conhecido" é transformado pela heurística transformacional de implementação 03 (vide Capítulo 3 – Tabela 3.2) na meta "numero_de_citacoes_por_ publicacao_seja_conhecido" do tipo atingir, Figura 7.13. O rastro é fornecido por essa mesma heurística, utilizando a conversão padronizada de nomes. Essa meta possui a *tag* exported="true", uma vez que deve ser ativada pelo agente que utiliza a capacidade. A Figura 7.13 também mostra o meta-*goal* que é ativado toda vez

que a meta está ativa, pois o JADEX ativa meta-goals quando existem dois ou mais planos diferentes para se atingir a mesma meta. O rastro para o meta-goal é oferecido pela heurística transformacional de implementação 15 (Tabela 3.2).

O meta-*goal* "metagoal1" dispara o meta-plano "metaplan1", esse último descrito na Figura 7.14.

```
<goals>
    <achievegoal name="numero_de_citacoes_por_publicacao_seja_conhecido"
                 exported="true"/>

    <metagoal name="metagoal1" recalculate="false">
        <parameterset name="applicables" class="ICandidateInfo"/>
        <parameterset name="result" class="ICandidateInfo" direction="out"/>
        <trigger>
            <goal ref="numero_de_citacoes_por_publicacao_seja_conhecido"/>
        </trigger>
    </metagoal>
</goals>
```

Figura 7.13 - As metas da capacidade "FornecedorDeCitacoes"

A Figura 7.14 mostra os planos da capacidade "FornecedorDeCitacoes", produzidos pelas heurísticas transformacionais a partir das intenções "Requisitar o número de citações para o Google Scholar" e "Delegar a meta a um agente móvel", apresentadas no modelo SR da Figura 7.8. A heurística transformacional de implementação 07 (vide Capítulo 3 – Tabela 3.2) oferece os rastros para os dois planos. Cada um dos planos foi nomeado de acordo com o padrão estabelecido pelo *framework* JADEX, e os rastros seguem essa conversão padronizada. Como pode ser observado através da *tag* "trigger", ambos os planos são disparados pela mesma meta.

O meta-plano "metaplan1" executa o método "body()" da classe Java "Metaplan", cujo *import* pode ser observado na Figura 7.11.

O plano "requisitar_o_numero_de_citacoes_para_o_google_scholar" instancia a classe "RequisitarONumeroDeCitacoesParaOGoogleScholar", apresentada nas Figuras 7.15 e 7.16. Essa classe estende a classe "Plan" oferecida pelo *framework* JADEX. Quando o plano precisa ser executado, o JADEX utiliza o método "body()". O rastro entre as classes Java e a especificação BDI é oferecido pela heurística transformacional de implementação 08 (vide Capítulo 3 – Tabela 3.2).

O corpo do método "body()" é preenchido pelas heurísticas copiando o cenário contido na *tag* "script" da intenção. O rastro desse processo é oferecido

pela heurística transformacional de implementação 09 (vide Capítulo 3 – Tabela 3.2). Na Figura 7.15, pode ser observado o cenário da Figura 7.10. Se o cenário inserido como comentários no corpo do método “body()” utilizar sub-cenários, esses sub-cenários devem ser implementados como métodos a serem chamados pelo método “body()”.

```
<plans>
    <plan name="requisitar_o_numero_de_citacoes_para_o_google_scholar">
        <body>new RequisitarONumeroDeCitacoesParaOGoogleScholar()</body>
        <trigger>
            <goal ref="numero_de_citacoes_por_publicacao_seja_conhecido"/>
        </trigger>
    </plan>

    <plan name="delegar_a_meta_a_um_agente_movel">
        <body>new DelegarAMetaAUmAgenteMovel()</body>
        <trigger>
            <goal ref="numero_de_citacoes_por_publicacao_seja_conhecido"/>
        </trigger>
    </plan>

    <plan name="metaplan1">
        <body class="MetaPlan"/>
        <trigger>
            <goal ref="metagoal1"/>
        </trigger>
    </plan>
</plans>
```

Figura 7.14 - Os planos da capacidade “FornecedorDeCitacoes”

```
package lattesscholar.sma.fornecedordecitacoes;

import jadex.runtime.Plan;

public class RequisitarONumeroDeCitacoesParaOGoogleScholar extends Plan {

    public void construirAExpressaoDeBusca() {...}

    public void construirAURLDeBuscaAPartirDaExpressao() {...}

    @Override
    public void body() {
        //Título: Requisitar o número de citações para o Google Scholar
        //Atores: Papel Fornecedor de citações, Agente Google Scholar
        //Sub-cenários: Construir a expressão de busca,
        //              Construir a URL de busca a partir da expressão
        //Meta: Requisitar o número de citações para o Google Scholar
        //Metas flexíveis impactadas: Performance,
        //          Evitar muitas requisições [Google Scholar]
        //Recursos: lista de publicações, publicação, número de citações,
        //          expressão de busca, URL de busca.
        //Episódios:
```

Figura 7.15 - O plano “RequisitarONumeroDeCitacoesParaOGoogleScholar” anotado com cenários

O engenheiro de software deve implementar cada cenário escrevendo o código de cada episódio do cenário logo abaixo do comentário correspondente. A Figura 7.16 mostra os episódios 2 e 3 já implementados como chamadas aos métodos dos sub-cenários.

```
//Episódios:
//1-Para cada publicação da lista de publicações:
...
//2-Construir a expressão de busca
this.construirAExpressaoDeBusca();
//3-Construir a URL de busca a partir da expressão
this.construirAURLDeBuscaAPartirDaExpressao();
//4-Fazer uma requisição HTTP ao Google Scholar usando a URL
//  de busca
...
//5-Recuperar o código fonte (HTML) da página Web de resposta
...
//6-Filtrar o número de citações do código em HTML
...
//7-Somar o número de citações caso encontre duas ou mais
//  instâncias da mesma publicação na página Web de resposta
...
```

Figura 7.16 - Implementando os episódios entre os comentários

7.2.4. Atividades Armazenar e Consultar

Os artefatos produzidos nas atividades Argumentar, Desenhar e Implementar, os modelos ITrace que fazem a pré-rastreabilidade de cada um desses artefatos, bem como os protótipos de interface discutidos na terceira reunião são armazenados para, futuramente, serem consultados.

Em linhas gerais, as atividades Armazenar e Consultar manipulam modelos SA, SD e SR do *framework* i*, grafos de argumentação na linguagem ACE, especificações BDI, cenários, modelos ITrace e, se necessário, diagramas UML que representam detalhes não capturados pelos modelos i*, como diagramas de sequencia e de estados.

Para garantir o livre acesso de todos os participantes do projeto aos artefatos produzidos nas várias iterações, foi criado um Wiki do projeto. Definimos uma estrutura hierárquica/cronológica padrão a ser seguida em todos os projetos. A Figura 7.17 mostra a estrutura proposta para o Wiki.

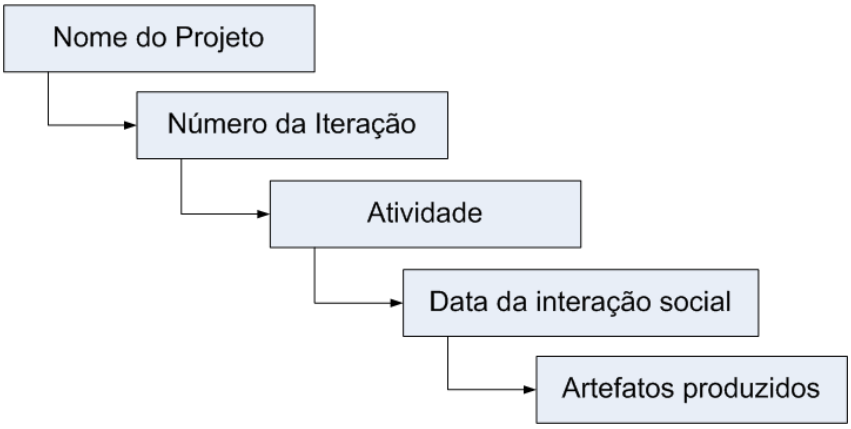

Figura 7.17 - A estrutura hierárquica/cronológica utilizada no Wiki do projeto

O Wiki do projeto é organizado de forma hierárquica, onde a página do projeto contém elos para as páginas das seis iterações. Estas, por sua vez, possuem elos para as três atividades do processo: Argumentar, Desenhar e Implementar. A página de cada atividade mantém uma lista cronológica de elos para as interações sociais. Cada interação social possui os artefatos produzidos na mesma, bem como o modelo ITrace que a modela.

Na atividade Desenhar, o Wiki oferece um elo para a seção de cenários da ferramenta C&L visando facilitar o acesso aos cenários produzidos nessa atividade. O Wiki também oferece um elo para a ferramenta de controle de versão dos artefatos.

7.3. Considerações Finais

Nesse capítulo, apresentamos uma abordagem que integra nossos trabalhos em relação a quatro desafios para a Transparência de Software, apresentados nos Capítulos 3 a 6, respectivamente: (i) anexar os requisitos ao código; (ii) trabalhar com modelos e análise de metas flexíveis em tempo de execução; (iii) elicitar e validar requisitos de transparência; e (iv) anexar modelos de pré-rastreabilidade ITrace a todos os artefatos produzidos no Processo de Desenvolvimento de Software Transparente.

A abordagem proposta baseia-se em um processo iterativo que possui cinco atividades, sendo três atividades principais e duas atividades que provêem retro-alimentação entre as iterações. As atividades principais são: Argumentar, Desenhar e Implementar. Cada uma das três atividades principais aplica pelo

menos dois dos trabalhos relacionados aos quatro desafios. A atividade Argumentar, por exemplo, aplica argumentação para a elicitação e validação relativa de requisitos de transparência, e aplica modelos de pré-rastreabilidade ITrace para se rastrear os processos que produzem os artefatos durante as reuniões.

Resumidamente, os requisitos de transparência são elicitados em discussões entre os interessados, que contam com o apoio do Catálogo de Transparência de Software para elaborar seus argumentos. Os grafos que representam esses argumentos são analisados por algoritmos da linguagem ACE. Se um consenso foi obtido, os requisitos de transparência são considerados **relativamente** validados. O uso de heurísticas para guiar o desenho e a implementação do software a partir desses requisitos relativamente validados visa garantir que essa validação relativa não seja perdida durante o processo de se baixar o nível de abstração dos artefatos do software. Todo trecho de código pode ser rastreado diretamente às abstrações do modelo i* ou a episódios de cenários que descrevem as tarefas.

Os artefatos produzidos em cada atividade principal são armazenados pela atividade Armazenar no Wiki do projeto, tornando esses artefatos disponíveis para todos os participantes. Esses artefatos podem ser consultados pelas atividades principais nas iterações seguintes através da atividade Consultar, fechando, em conjunto com a atividade Armazenar, um ciclo que permite a retro-alimentação.

Outros trabalhos visam à implementação de SMAs a partir de requisitos intencionais modelados no *framework* i*. O principal deles é a metodologia Tropos (Castro 2002; Bertolini 2006), que oferece um desenvolvimento dirigido por modelos. Entretanto, desenvolver SMAs a partir de requisitos intencionais em um processo dirigido por modelos não é suficiente para que se diga que o software produzido é transparente. Nossa abordagem, por exemplo, elicita e valida relativamente requisitos de transparência através do uso da técnica de argumentação, o que nos permite dizer que os requisitos incluem os requisitos de transparência com base nos pontos de vista dos interessados. Assim, nossa abordagem integra nossos esforços em lidar com vários desafios e em níveis de abstração diferentes para poder afirmar que, ao final do processo, o software é *suficientemente* transparente.