Aplicação de Modelos Intencionais em Sistemas Multiagentes para Estabelecer Políticas de Monitoração de Transparência de *Software*

André Luiz de Castro Leal [1,2]

Henrique Prado Sousa [1]

Julio Cesar Sampaio do Prado Leite [1]

Carlos José Pereira Lucena [1]

Resumo. Transparência é fator importante nas relações do cotidiano empresarial moderno e dos cidadãos e a percepção de tal qualidade sugere confiança. As aplicações computacionais inseridas nesse contexto desafiam engenheiros de *software* a entender e estabelecer políticas de transparência como característica desses sistemas. Como requisito não funcional, ela necessita de técnicas, métodos e ferramentas que proporcionem sua licitação, modelagem, como também validação. O presente trabalho propõe a modelagem intencional para estabelecer políticas de monitoração a partir de agentes monitores autônomos. As políticas são executadas e validadas em um *software* independente com o objetivo de verificar conformidades a partir de um catálogo normatizado com atributos de transparência.

Abstract. *Transparency is an important factor in relations between modern business and citizens, since the perception of this quality suggests. The computer applications included in this context challenge software engineers to understand and establish policies able to insert transparency as characteristic of these systems. As non-functional requirement, it requires techniques, methods and tools that provide its elicitation, modeling, as well as validation. This paper proposes the establishment of monitoring policies from autonomous monitors agents designed with the support of intentional modeling. The policies are applied and validated in an independent software with the objective of verifying compliance from standardized catalog composed with attributes of transparency.*

1 Departamento de Informática, PontifíciaUniversidade Católica do Rio de Janeiro
{aleal, hsousa, julio, lucena}@inf.puc-rio.br
2 Departamento de Matemática, Universidade Federal Rural do Rio de Janeiro (UFRRJ)

1 Introdução

O termo transparência tem sido amplamente utilizado atualmente nos diversos setores da sociedade. Esse fato é constatado a partir da exploração de tal característica pela mídia, pelas campanhas governamentais, nas propagandas empresariais, no ambiente corporativo e no discurso de candidatos a cargos políticos. Tal evento é principalmente impulsionado pelo anseio atual da população pelo tema, uma vez que deseja ser informada e de ter acesso à informação de maneira apropriada [4].

A transparência poderá auxiliar decisões se proporcionar às pessoas informações pertinentes sobre determinado contexto analisado [1] e inclusive alterar posições de poder a partir de informações disponíveis, corretas e abertas [2]. Segundo Holzner [2], o acesso à informação permite criar uma sociedade democrática com cidadãos participativos e capazes de compreender e utilizar a informação que lhes é disponibilizada.

Dessa forma, transparência é vertente importante no contexto de mudanças locais e globais da sociedade devido à complexidade das relações sociais e comerciais postas atualmente [3]. Negócios necessitam validar informação sobre mercados, seus riscos e oportunidades; políticas necessitam validar informação sobre intenções e estratégias entre países; empresas necessitam estabelecer relações comerciais e necessitam de transparência em seus processos e informação [2]; de modo geral a transparência entra nesse contexto como um adjetivo singular para demonstrar a seriedade nas relações interpessoais e corporativas [3].

A partir dessa realidade há uma necessidade de criação de sistemas transparentes que tenham por características cadeias de ações e respostas a partir do envolvimento de dois grupos de sujeitos: aqueles que provêm à informação e aqueles que recebem a informação produzida pelas políticas transparentes, ou seja, os usuários [1]. Portanto, sistemas transparentes necessitam de dois contextos: o processo de disponibilização da informação e a informação em si [10].

Diante disso, transparência é uma característica muito bem vista para instituições e para os cidadãos. No entanto, temos uma vertente importante nesse preâmbulo que é o *software*, pois o mesmo permeia vários aspectos da nossa sociedade, e em algum ponto no futuro, engenheiros de *software* terão que corresponder a mais uma demanda de requisito: Transparência [4]. Neste ambiente vislumbrado, engenheiros terão que possuir métodos, técnicas e ferramentas para ajudar a construir *software* transparente [4].

O Grupo de Pesquisas em Engenharia de Requisitos da PUC-RIO (Grupo ER - PUC-Rio) [5] trabalha na elaboração, validação e evolução de um catálogo de transparência de *Software* (TS) [6]. O grupo entende que TS está relacionada com a característica de transparência aplicada nos diversos contextos de *software*, por exemplo, em sua concepção, em sua execução, na geração de informações, na informação em si, em seus modelos, nas relações humanas que conduzem o processo de desenvolvimento e em artefatos de forma geral. Para isso registra no catálogo de TS as definições de atributos, grupos, questões e alternativas que podem

auxiliar na compreensão ou podem potencializar a TS [6]. Diante da nova perspectiva de demanda da sociedade e dos desafios dos engenheiros de *software* em tratar o requisito de transparência, há uma frente de pesquisa ainda pouco explorada no cenário nacional e internacional e que desperta interesse por parte do Grupo ER - PUC-RIO: a monitoração de TS.

A partir desse contexto, surgiu a intenção de explorar técnicas, métodos e procedimentos para monitoração da TS. Um primeiro esforço é a modelagem intencional de agentes monitores, bem como sua implementação baseada em sistemas multiagentes (SMA) [7] [8], para que esses agentes sejam capazes de efetuar coleta de dados confiáveis em tempo real sobre determinado processo de *software*. A partir disso, tratar a divulgação de conformidades e não conformidades do *software* monitorado de acordo com padrões do catálogo de TS.

A modelagem intencional permite modelar contextos organizacionais baseado nos relacionamentos de dependência intencional entre os diversos atores e suas interações [19]. O foco da modelagem é possibilitar a compreensão das intenções internas e àquelas compartilhadas entre os atores, uma vez que as mesmas são expressas explicitamente. O *framework* i* de Yu [19] é utilizado para a criação dos modelos intencionais.

Outras duas características são associadas ao processo de monitoração por agentes: primeiramente a questão da confiança (*trust*) [37], que está relacionado na questão de acreditar ou se ter confiança em alguém ou alguma coisa; e a estratégia de *provenance* [27] [28] [31], que tem sido utilizado como processo para manter rastros de dados para a obtenção da forma com que esse dado foi produzido e para a verificação da qualidade do processo que o produziu.

O uso dessas diversas abordagens traz um desafio face ao pouco conhecimento de sua interação. A prova de conceito é aplicada sob os traços de execução do *software Lattes Scholar* (LS) [13], que será melhor caracterizado posteriormente no texto.

O artigo está subdividido em seções onde a seção 2 apresenta uma visão sobre TS, seus graus e os *patterns*; a seção 3 apresenta a proposta do modelo intencional nas relações entre os agentes; a seção 4 aborda sobre os modelos canônicos e a estratégia de *provenance* utilizados no processo de monitoração do LS; na seção 5 estão descritas as questões sobre implementação e por fim, na seção 6 as conclusões.

2 Transparência de *software*

A transparência, por ser um requisito não funcional [4], torna-se uma característica não mensurável, podendo somente ser definida como satisfatória a partir do ponto de vista de um indivíduo, ou seja, um produto pode ser considerado transparente na opinião de um indivíduo enquanto de outro não. Portanto, cada esforço realizado em um dado produto com o objetivo de torná-lo transparente, apenas pode ser considerado como uma atividade que auxiliará no incremento do nível de transparência, tornando-o mais transparente, uma vez que não é possível definir se um produto é de fato totalmente transparente devido às suas características não funcionais.

Cappelli et al. [9] define um conjunto de características que podem contribuir com transparência, conceituadas pela autora como Graus de Transparência (GT). São cinco graus na camada mais alta de uma hierarquia que auxilia a maturidade em transparência, que são denominados: a) Acessibilidade: a transparência é realizada através da capacidade de acesso. Esta capacidade é identificada através da aferição de práticas que implementam características de portabilidade, operabilidade, disponibilidade, divulgação e desempenho; b) Usabilidade: a transparência é realizada através das facilidades de uso. Esta capacidade é identificada através da aferição de práticas que implementam características de uniformidade, intuitividade, simplicidade, amigabilidade e compreensibilidade; c) Informativo: a transparência é realizada através da qualidade da informação. Esta capacidade é identificada através da aferição de práticas que implementam características de clareza, acurácia, completeza, corretude, consistência e integridade; d) Entendimento: a transparência é realizada através do entendimento. Esta capacidade é identificada através da aferição de práticas que implementam características de composição, concisão, divisibilidade, dependência, adaptabilidade e extensibilidade; e) Auditabilidade: a transparência é realizada através da auditabilidade. Esta capacidade é identificada através da aferição de práticas que implementam características de explicação, rastreabilidade, verificabilidade, validade e controlabilidade [10].

Cada grau de transparência possui um conjunto de atributos que são necessários para que aquele grau ofereça um bom nível de contribuição para a transparência. A Tabela 1 apresenta a relação discriminada desses atributos.

Tabela 1. Detalhamento dos atributos de GT. Fonte: [10]

Graus de transparência	Atributos	Descrição do atributo
Acessibilidade	Portabilidade	Capacidade de ser usado em diferentes ambientes.
Acessibilidade	Disponibilidade	Capacidade de ser utilizado no momento em que se fizer necessário.
Acessibilidade	Divulgação	Capacidade de ser apresentado.
Usabilidade	Uniformidade	Capacidade de manter uma única forma.
Usabilidade	Simplicidade	Capacidade de não apresentar dificuldades ou obstáculos.
Usabilidade	Operabilidade	Capacidade de estar operacional.
Usabilidade	Intuitividade	Capacidade de ser utilizado sem aprendizado prévio.
Usabilidade	Desempenho	Capacidade de operar adequadamente.
Usabilidade	Adaptabilidade	Capacidade de mudar de acordo com as circunstâncias e necessidades.
Usabilidade	Amigabilidade	Capacidade de utilização sem esforço
Informativo	Clareza	Capacidade de nitidez e compreensão.
Informativo	Completeza	Capacidade de não faltar nada do que pode ou deve ter.
Informativo	Corretude	Capacidade de ser isento de erros.
Informativo	Atualidade	Capacidade de estar no estado atual.
Informativo	Comparabilidade	Capacidade de ser comparado.

Informativo	Consistência	Capacidade de resultado aproximado de várias medições de um mesmo item.
Informativo	Integridade	Capacidade de correto e imparcial.
Informativo	Acurácia	Capacidade de execução isenta de erros sistemáticos.
Entendimento	Concisão	Capacidade de ser resumido.
Entendimento	Compositividade	Capacidade de construir ou formar a partir de diferentes pares.
Entendimento	Divisibilidade	Capacidade de ser particionado.
Entendimento	Detalhamento	Capacidade de descrever em minúcias.
Entendimento	Dependência	Capacidade de identificar a relação entre as partes de um todo.
Auditabilidade	Validável	Capacidade de ser testado por experimento ou observação para identificar se o que está sendo feito é correto.
Auditabilidade	Controlabilidade	Capacidade de domínio.
Auditabilidade	Verificabilidade	Capacidade de identificar se o que está sendo feito é o que deve ser feito.
Auditabilidade	Rastreabilidade	Capacidade de seguir o desenvolvimento de um processo ou a construção de uma informação, suas mudanças e justificativas.
Auditabilidade	Explicável	Capacidade de informar a razão de algo.

Os graus e atributos possuem a capacidade de interação uns com os outros, sendo capazes de influenciar de forma positiva ou negativa. Capelli [10] apresenta um grafo de transparência desenvolvido a partir do *Non-Funcional Requirements Framework*, proposto em [11]. Este *framework* define uma forma sistemática para decompor requisitos não funcionais (características de qualidade), priorizar, operacionalizar e tratar interdependências entre eles, independentemente de quais sejam.

Nesta estrutura, também são representados os tipos de contribuição que os elementos podem ter em sua relação. Os tipos representam: a) *Break*: provê contribuição negativa suficiente para que a característica superior não seja atendida; b) *Hurt*: provê contribuição negativa parcial para não atendimento da característica superior; c) *Unknown*: provê contribuição porém não se sabe se negativa ou positiva; d) *Help*: provê contribuição positiva parcial para atendimento da característica superior; e) *Make*: provê contribuição positiva suficiente para que a característica superior seja atendida [10]. A Figura 1 apresenta os tipos de contribuição, atribuídos dos GT e suas relações. Estão representados somente os tipos *Help* e *Make*.

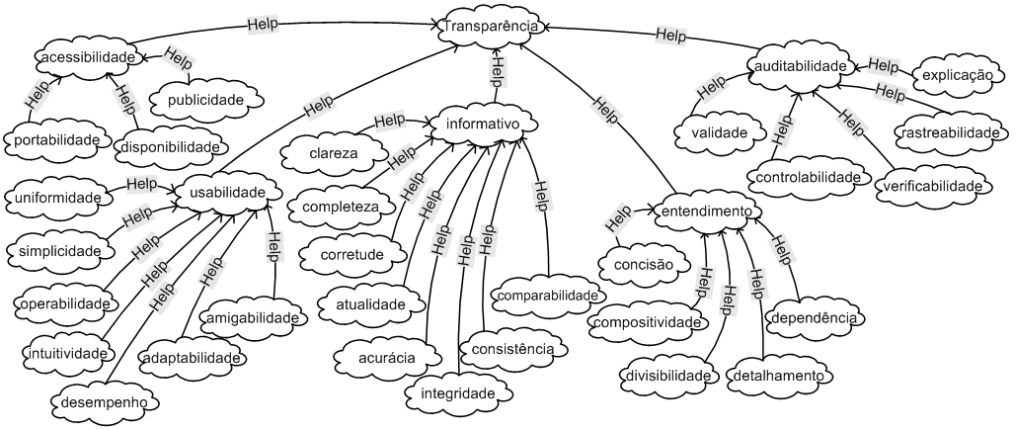

Figura 1. Diagrama de relacionamento de atributos de transparência. Fonte: [10]

2.1 *Patterns* de transparência

O Grupo ER - PUC-Rio, em sua linha de pesquisa de TS [6], vem trabalhando na evolução do catálogo de transparência [12]. O catalogo está dividido em camadas, sendo que incialmente são descritas metas ou objetivos de transparência no qual chamamos de atributos. A partir dos atributos, os outros elementos foram subdivididos em grupos, questões e alternativas, esses itens foram denominados como *patterns* de transparência [12].

A criação dos *patterns* foi baseada na adaptação do método Goal-Question-Metrics (GQM) [40] para uma abordagem denominada Goal-Question-Operacionalization (GQO) [39]. O método GQM auxilia na definição de medidas, a partir de um sistema de medição, para desenvolvimento de *software*. Ele inicia com objetivos para o *software*, define questões para avaliar a satisfação com relação aos objetivos e cria métricas que permitem que essas questões sejam respondidas qualitativamente. Na adaptação do método para o GQO, métricas são alteradas para respostas para operacionalizações.

O GQO é baseado na abordagem de boas práticas de *software*, na qual questions são respondidas a partir da identificação de práticas de engenharia de software que operacionalizam as metas ou objetivos do catalogo de transparência. Dessa forma, as operacionalizações são tratadas no *pattern* como alternativas. Os elementos grupos, questões e alternativas foram criados segundo critérios definidos a partir das boas práticas e escolhidos segundo um debate amplo dos membros do grupo e também validados por outros pesquisadores da área. A completude do catalogo pode ser consultada em [12].

Como exemplo, pode-se representar o atributo Rastreabilidade. Seus grupos seriam: fazer pré-rastreabilidade, fazer rastreabilidade em tempo de desenho e fazer rastreabilidade em tempo de execução. Cada grupo possui seu conjunto de questões e as questões têm seu conjunto de alternativas.

As alternativas relacionadas a cada questão são opções de verificação de sua conformidade, ou seja, dada uma questão, suas alternativas servem como parâmetro de verificação de respostas positivas ou negativas. Isso também ocorre para a verificação dos grupos a partir de suas questões positivas.

A Tabela 2 apresenta a lista de grupos, questões e alternativas estabelecidas pelo Grupo ER - PUC-Rio para compor os *patterns* de transparência do atributo Rastreabilidade. Grupos, questões e alternativas estão identificados pelas letras G, Q e A, respectivamente.

Tabela 2. Grupos, questões e alternativas sobre rastreabilidade de transparência.

Rastreabilidade: Grupos / Questões / Alternativas	Clas.
1. Fazer Pré-Rastreabilidade.	G
1.1 As fontes de informação utilizadas são rastreadas?	Q
1.1.1 São identificados atores que passam a informação.	A
1.1.2 É identificado o momento em que a informação foi coletada.	A
1.2 Os recursos utilizados ou que serão necessários são rastreados?	Q
1.2.1 Identificação de erros (*error*...), cabeçalhos de *Logs* (resumo, objetivo, justificativa, número de linhas geradas).	A
1.3 As interações entre interessados são rastreadas?	Q
1.3.1 Registros de troca de mensagens.	A
1.4 As principais metas e preocupações dos *stakeholders* são rastreadas?	Q
1.5 Os impactos sociais foram explicitados?	Q
1.5.1 Em modelos intencionais: as condutas, configuração de relações (*means end*, *sequencial task*), identificação das interações de funções a partir de comentários.	A
1.6 As redes de interação são mapeadas?	Q
2 Fazer Rastreabilidade em tempo de desenho.	G
2.1 Os rastros entre diferentes artefatos da Engenharia de *Software* são mantidos?	Q
2.2 Os rastros entre diferentes versões de um mesmo artefato são mantidos?	Q
2.3 Os raciocínios *(rationales)* das tomadas de decisão são rastreados?	Q
2.4 As questões sociais em tempo de desenho são rastreadas?	Q
2.5 As redes de interação são mapeadas?	Q
2.5.1 O nome dos agentes envolvidos nas comunicações, nome do *software* e sistemas externos, serviços (*webServices*) e o registro de suas comunicações, passagem de parâmetros, troca mensagens.	A
3. Fazer Rastreabilidade em tempo de execução.	G
3.1 São anotadas as fontes dos dados?	Q
3.1.1 Endereços web de pesquisa, nomes de servidores, agentes, *software*, processos.	A
3.2 São rastreadas as modificações/atualizações dos dados?	Q
3.3 São mantidos os rastros dos processos executados/aplicados?	Q
3.3.1 Início e fim de execuções, *URL* registradas, nome do agente, *software*	A

executado, portas, servidores, páginas web rastreadas, resultados obtidos no rastreamento.
3.4 Os interessados têm acesso aos registros de rastreabilidade? Q

3 Modelo intencional dos agentes monitores

O primeiro esforço de análise de viabilidade de agentes monitores na detecção de conformidades ou não com o catálogo de transparência foi efetuado sob os traços de execução gerados a partir de agentes desenvolvidos na arquitetura do LS [13]. O LS é um SMA independente para contagem de citações científicas, que combina os serviços do sítio do *Lattes* [14] e do *Google Scholar* [15]. Sua execução é baseada na pesquisa inicial pelo nome do autor no sítio do *Lattes* e posteriormente uma pesquisa pela relação de artigos no *Google Scholar* com o objetivo de contabilizar citações sobre cada publicação do autor. O resumo do funcionamento do LS pode ser visto na Figura 2.

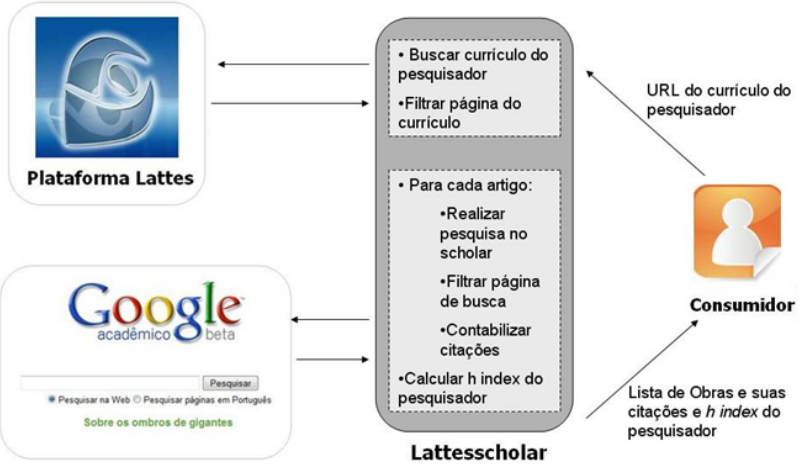

Figura 2. Funcionamento do LS.

A vantagem de seu uso está justamente na pesquisa pelas citações a partir do título da publicação, o que difere, por exemplo do *Publish and Perish* [16], do *Microsoft Academic Search* [17] e do *Google Scholar Citation* [18], uma vez que suas pesquisas são realizadas a partir do nome do autor podendo acarretar inconsistências de contagem devido a erros causados por homônimos, abreviaturas, nomes muito extensos e sinônimos em nomes muito pequenos.

A partir da identificação do *software* a ser monitorado foi elaborado um modelo intencional baseado em i* (i-estrela) [19]. A escolha da abordagem i* é justificada devido aos benefícios da proposta do *framework* de Yu [19], que sugere a modelagem de contextos organizacionais baseado nos relacionamentos de dependências entre atores; é usado para obter

um melhor entendimento dos relacionamentos; possibilita a compreensão das razões internas dos atores, uma vez que as mesmas são expressas de forma explícita; e auxilia na escolha de alternativas durante a etapa de modelagem do *software*. A identificação destes relacionamentos auxiliou na elicitação das metas dos agentes em seus comportamentos internos e externos. Os atores ou agentes são entidades ativas que efetuam ações para alcançar metas através do exercício de suas habilidades e conhecimentos, podendo ter dependências intencionais entre eles. Uma dependência intencional ocorre quando dois ou mais atores compartilham algum elemento de dependência [20].

Os elementos do *framework* utilizados no modelo de monitoração são os definidos por Yu [19] e podem ser vistos de forma resumida e organizada nas Figuras 3 e 4.

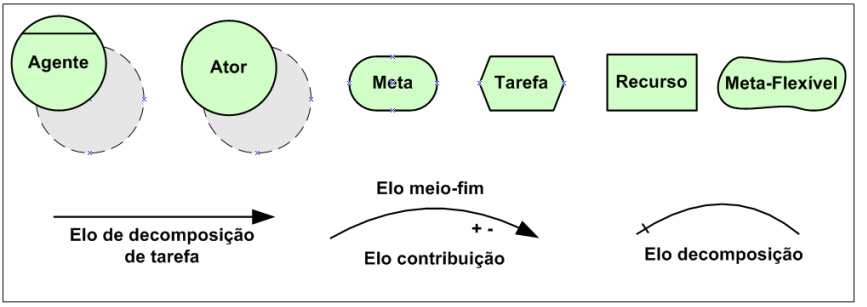

Figura 3. Elementos i* de modelagem intencional. Fonte: [19].

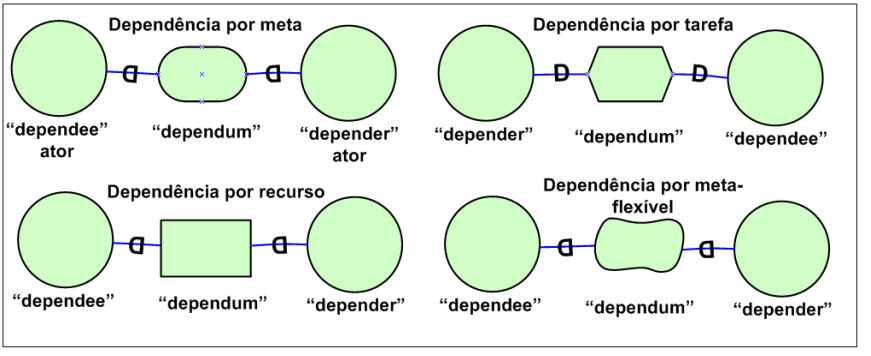

Figura 4. Tipos de relacionamentos. Fonte: [19]

As declarações explícitas nesse tipo de modelo permitem uma melhor avaliação das intenções internas dos atores, bem como naquelas compartilhadas com outros envolvidos. O modelo foi elaborado com a preocupação de se manter a abordagem de estados mentais dos agentes compostos por crença (*Belief*), desejo (*Desire*) e intenção (*Intention*), denominado na literatura de arquitetura BDI [21] [22], além de uma noção de planos (*Plans*) e objetivos (*Goals*)

[21], [22], [23], [24], [25] e [26]. Dessa forma, pretende-se garantir um modelo mais aderente à proposta de monitoração de alternativas a partir de uma abordagem baseada em SMA e potencializa a escalabilidade e evolução para futuras pesquisas.

A interação dos agentes é determinada pelo modelo intencional conforme Figura 5:

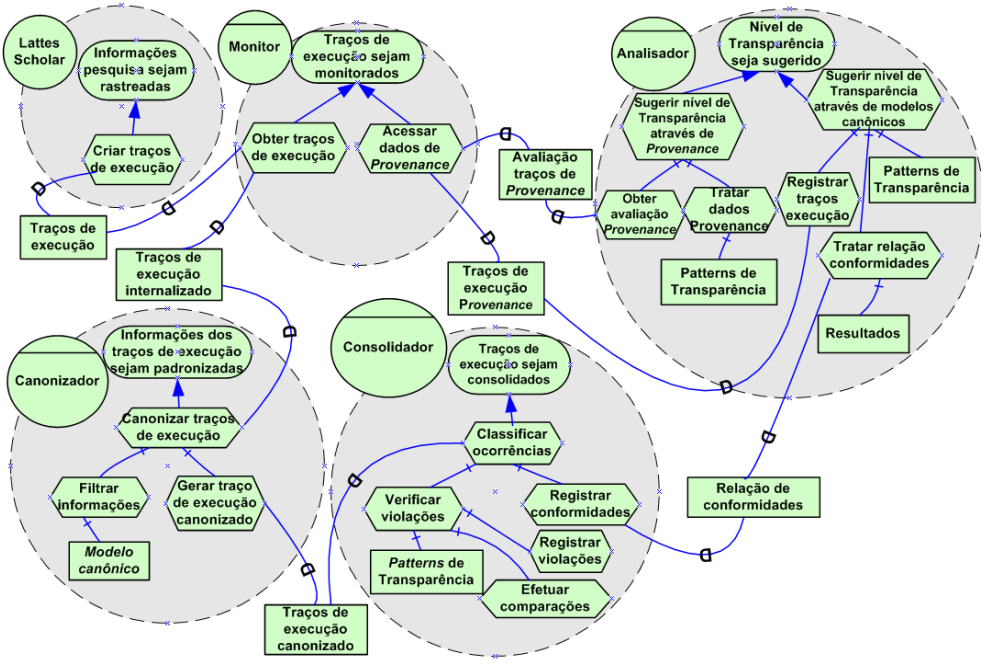

Figura 5. Modelo intencional da interação dos agentes monitores – *framework i** [19].

O modelo intencional apresenta os atores envolvidos no processo de monitoração. Pode-se verificar a presença do LS e os quatro agentes de monitoração. Os quatro agentes no SMA descritos no modelo atuam sobre os traços de execução do LS.

Os traços de execução do LS são registros armazenados a cada iteração de execução do LS a partir de pesquisas por nome do autor na plataforma *Lattes* e o seu número de citações a partir do *Google Scholar*, e seu posterior arquivamento. Os traços são armazenados de acordo com o exemplo. O exemplo não representa a completude dos traços do LS devido ao extenso tamanho do original.

Os traços de execução possuem registros de cada iteração do LS a partir do armazenamento das ações dos agentes, como *ReceberUrlPesquisador*, *SolicitarObrasCurriculo*, entre outras. Além disso, conforme apresentado na linha 20, os traços contém o número de citações por obra do autor. Para reforçar o explicado anteriormente, o LS é um software independente do sistema de monitoração e dele são utilizados apenas os traços de sua execução.

1.	20100913 15:43:00: state=2
2.	20100913 15:43:01: behaviour type=istar.behaviour.SequentialTaskBehaviour name=ObterCitacoesPesquisador adding subBehaviour
3.	20100913 15:43:01: behaviour type=basicelement.ReceberUrlPesquisador$myBehaviour name=ReceberUrlPesquisador
4.	20100913 15:43:01: behaviour type=istar.behaviour.SequentialTaskBehaviour name=SolicitarObrasCurriculo adding subBehaviour
5.	20100913 15:43:01: behaviour type=istar.behaviour.RequestExternalElement name=Obras
6.	20100913 15:43:01: behaviour type=istar.behaviour.SequentialTaskBehaviour name=SolicitarObrasCurriculo
7.	20100913 15:43:01: behaviour type=istar.behaviour.SequentialTaskBehaviour name=SolicitarCitacoesObras adding subBehaviour
8.	20100913 15:43:01: behaviour type=istar.behaviour.RequestExternalElement name=Citacoes
9.	20100913 15:43:01: behaviour type=istar.behaviour.SequentialTaskBehaviour
10.	20100913 15:43:01: behaviour type=class maingoal.CitacoesObrasObtidas$ObterCitacoes name=Initial state=READY
11.	20100913 15:45:24: Agent name=manager1@VIVALDI:1099/JADE sending message manager1@VIVALDI:1099/JADE_1 to agent ed@VIVALDI:1099/JADE
12.	20100913 15:45:27: Message content: Viewpoint analysis: a case study.
13.	20100913 15:45:27: Message content: Reutilizacao de Software..
14.	20100913 15:45:27: Message content: SADT Datagrams, A Powerful Tool for Requirements Analysis.
15.	20100913 15:45:27: Message content: O Enfoque Social na Análise de Sistemas.
16.	20100913 15:45:27: Message content: Requirements Engineering and Aspects.
17.	20100913 15:45:27: Message content: On Non-Functional Requirements in Software Engineering.
18.	20100913 15:45:27: Message content: Ontologias: Como e porque criá-las.
19.	20100913 15:45:27: Message content: Perspectives on Software Requirements: An Introduction.
20.	20100913 15:47:27: Agent name=manager1@VIVALDI:1099/JADE received a REPLY to manager1@VIVALDI:1099/JADE_1 with content:[O Uso do Paradigma Transformacional No Porte de Programas Cobol==2, Elicitacao de Requisitos==3, O Uso de Pontos de Vista Na

	Elicitacao de Requisitos.==0, Recovering business rules from structured analysis specifications==18, Draco-PUC: a technology assembly for domain oriented software development==63, An Elicitation Strategy Anchored on ISO 9000 Documents==0, Sistemas de Informação e Engenharia de Software, o Elo Gerencial==0, Lexicon Based Ontology Construction==12, Reutilizacao de Software.==1, Faes - Uma Estrategia Para Aquisicao de Informacoes==0,
21.	20100913 15:47:36: behaviour type=istar.behaviour.RequestExternalElement name=Citacoes status=done

Os traços de execução serão monitorados pelos agentes do SMA e as evidências encontradas serão comparadas ao catalogo de transparência para indicar conformidades e não conformidades com o mesmo. A Figura 5 deixa explícito que os quatro agentes no SMA possuem ações bem definidas e especializadas. Cada agente trata de forma distinta suas tarefas, recursos, desejos e objetivos. Suas discriminações são apresentadas como:

- MONITOR: o agente monitora constantemente os traços de execução gerados pelos agentes do LS, com o objetivo de captar os seus registros. Os registros dos traços de execução do LS são gerados pela própria aplicação, inicialmente ele verifica se já há traços de execução do agente ANALISADOR com resultados de avaliações. Para isso ele acessa um arquivo com o registro de rastros (traços de execução) do ANALISADOR referente à memória de avaliações anteriores, tais rastros são registrados a partir de uma estratégia de *provenance* [27] [28] [31]. O traço de execução do ANALISADOR, ou "memória de monitoração", contém o registro de *tags* das regras de monitoração baseados no modelo canônico e a correlação dessas *tags* com as alternativas do catálogo de TS de monitorações anteriores.

Dessa forma, o MONITOR decide por encaminhar diretamente ao ANALISADOR as respostas de conformidade de transparência sem ter a necessidade de transcorrer por todo o processo de monitoração, ou seja, passando por outros agentes como o CANONIZADOR e o CONSOLIDADOR.

Caso o agente não encontre os traços de execução dos rastros do ANALISADOR ele procede com o processo completo e envia uma mensagem ao agente CANONIZADOR informando sobre a captura e disponibilidade do arquivo.

- CANONIZADOR: tem por objetivo receber os registros dos traços de execução e transformar as informações em um padrão que possa ser consolidado. O CANONIZADOR utiliza endereços de posição nos traços de execução, como também ocorrências delimitadas por algum trecho de texto que evidencie o que está sendo canonizado. Por exemplo: o nome de um agente que está sendo monitorado pode ocorrer após a descrição *Agente name=*, inicia uma posição após o sinal de igual (=) e finaliza uma posição a frente do caractere @. Ou seja, o agente interpreta o arquivo dos traços de execução, padroniza sua leitura e disponibiliza os registros dos traços de execução em classes na memória, para que possam ser utilizadas por outros agentes.

- CONSOLIDADOR: verifica a partir da estrutura canonizada os registros de conformidades e não conformidades e disponibiliza ao ANALISADOR recursos de conformidades que possam ser verificados de acordo com o catálogo de transparência.

- ANALISADOR: tem por finalidade verificar se os registros padronizados e extraídos pelo CANONIZADOR correspondem às exigências dos Grupos, Questões e Alternativas normatizadas para um atributo de transparência. O agente verifica quais são as exigências da norma para que uma alternativa seja positivada. Por exemplo, para ser atendida uma alternativa para a questão: 2.5 As redes de interação são mapeadas? Alternativa: 2.5.1 o nome dos agentes envolvidos nas comunicações, nome do *software* e sistemas externos, serviços (*webServices*) e o registro de suas comunicações, passagem de parâmetros, troca mensagens, bastaria que o agente ANALISADOR encontrasse nos registros gerados pelo CANONIZADOR, o registro do nome dos agentes ou de um agente envolvido na troca de mensagens de *Send* e *Receive*. Assim o agente deve marcar a alternativa como satisfeita, ou positiva. O ANALISADOR guarda o traço de execução com o objetivo de manter uma memória das *tags* avaliadas nos *software* monitorados e sua correlação com o catálogo de transparência.

4 Modelo canônico e estratégias de *provenance*

A estratégia de monitoração foi aplicada sobre um modelo normatizado com as regras de leitura dos traços de execução do LS para orientar a execução e interação dos agentes, tal modelo foi denominado modelo canônico. Com o uso do modelo é possível implementar os agentes monitores para ler os registros dos traços de execução do *software* monitorado e transformá-lo em um modelo normatizado para ser comparado aos *patterns* de transparência.

A normatização foi definida a partir de uma abordagem 5w2h onde os quesitos foram definidos como: *What*: o que será feito (etapas); *Where*: onde será feito (local) – De/Para; *Why*: por que será feito (justificativa); *When*: quando será feito (tempo) – Data/Hora da execução dos processos; *Who*: por quem será feito (responsabilidade) – Agente responsável; *How*: como será feito (método) – Detalhamento das atividades do processo: Solicitação, Recebimento...; *How much*: quanto custará fazer (custo) – tempo de execução dos processos ou relativo a alguma exceção.

A monitoração com base no 5w2h permitirá ao agente CANONIZADOR criar um traço de execução canonizado que permite ao CONSOLIDADOR verificar as ocorrências e classificá-las em conformidades e não conformidades. Estamos considerando como conformidades, os registros dos traços de execução do *software* monitorado que estiverem de acordo com as regras 5w2h estabelecidas no modelo canônico, e as não conformidades são as violações às regras descritas. A Tabela 3 apresenta a proposta de modelo canônico utilizado para a formatação dos arquivos dos traços de execução do LS.

Tabela 3. Modelo canônico (regras de monitoração).

MONITORAÇÃO (*what*) - Rastreabilidade (*what*) - TS (*why*) Fonte (*where*): traços de execução *Lattes Scholar*: arquivo lattes1-log-2010-09-13 15-43-00.txt Local de Armazenagem (where): //SERVIDOR/C/PASTA/ Tipo: Arquivo ASCII			
O que monitorar? (*what*)	Posição no arquivo? (*where*) (coluna - linha) ou Quando ocorre? (*when*)	Formato	Exceção (*how much*)
Data de registro no traço de execução	1 a 8, registrada em todas as linhas.	AAAAMMDD	Não ocorre quando o registro anterior é continuado na linha *posterior*
Hora de registro no traço de execução	9 a 18, registrada em todas as linhas	HH:MM:SS	Não ocorre quando o registro anterior é continuado na linha posterior
Nome do agente utilizado pelo *Software*	Ocorre após a descrição "Agente name=". Inicia uma posição após o sinal de = e finaliza uma posição a frente do sinal @	Minúsculo, pode conter caracteres alfanuméricos	Há ocorrências após a descrição agent= que deve ser desprezada.
Nome do servidor onde o Agente é executado	Ocorre após o @ do nome do agente e é finalizado por : (dois pontos)	Maiúsculo, pode conter caracteres alfanuméricos	Não se aplica
Porta em que é executado	Ocorre após os : (dois pontos) do nome do servidor e é finalizado com / (barra simples), que não compõe o nome.	caracteres alfanuméricos	Não se aplica
Nome de recurso (padrão da notação i*) utilizado pelo Agente no *Software*	Ocorre após a palavra *Resource*. Inicia após o : (dois pontos) encontrado depois da palavra Resource e finaliza uma posição antes ao espaço.	Primeira letra Maiúscula, pode conter caracteres alfanuméricos	Não se aplica
Nome de páginas Web registradas	Ocorre após o *caractere* & e escrita até o espaço em branco, ou enter para outra linha.	caracteres alfanuméricos	Não se aplica
Há registro da troca de mensagens entre os agentes, considerando o nome dos agentes comunicadores?	20100913 15:45:18: Agent name=manager1@VIVALDI:1099/JADE sending message manager1@VIVALDI:1099/JADE_1 to agent lattes1@VIVALDI:1099/JADE	caracteres alfanuméricos	Não se aplica
Há registro da	20100913 15:45:18: Agent	caracteres	Não se aplica

troca de mensagens entre os agentes, considerando qual agente envia e qual agente recebe a mensagem?	name=manager1@VIVALDI:1099/JADE sending message manager1@VIVALDI:1099/JADE_1 to agent lattes1@VIVALDI:1099/JADE	alfanuméricos	
Há registro do conteúdo das mensagens trocadas entre os agentes?	20100913 15:45:18: Message content: http://lattes.cnpq.br/6871006250321522	caracteres alfanuméricos	Não se aplica

A monitoração de transparência pelos agentes monitores ocorre com base em duas abordagens. A primeira é feita através da busca pelos *tags* 5w2h do modelo canônico nos traços de execução do *software* monitorado, no caso o LS, de forma sequencial e linha a linha. As evidências de conformidades e não conformidades, ou violações de regra, dos traços de execução do *software* com o catálogo de transparência são anotadas. Nesse momento da pesquisa estamos focando apenas as conformidades para efeito de identificação de transparência, as não conformidades são apenas registradas e serão utilizadas em trabalhos futuros. A segunda abordagem da monitoria é baseada na estratégia do uso de *provenance* [31].

4.1 Estratégia de *provenance* para memorizar rastros do agente analisador

Provenance tem sido utilizado por pesquisadores para manter rastros de dados para a obtenção da forma com que esses dados foram produzidos e para a verificação da qualidade do processo que o produziu [27] [28]. O dicionário Inglês *Oxford* [30] define *provenance* como: '*(i) The fact of coming from some particular source or quarter; origin, derivation; (ii) The history of the ownership of a work of art or an antique, used as a guide to authenticity or quality; a documented record of this*'.

Pinheiro, McGuinness and McCool [31] and Vasquez, Gomadam, and Peterson [33] sugerem que *provenance* tem um foco baseado na história, no processo e na transformação da origem dos dados, mas também deveria prover métricas qualitativas e quantitativas para analisar a qualidade e a segurança do dado baseado em *trust* (confiança) [37]. A palavra *trust* será utilizada ao longo do texto por seu significado é comum na literatura para expressar confiança. Com a abordagem pretende-se envolver aspectos de *trust* para influenciar a confiança sobre a origem da criação dos dados e sobre a origem de suas modificações.

O conceito de *trust* está relacionado na questão de acreditar ou se ter confiança em alguém ou alguma coisa [37]. O dicionário Inglês *Oxford* [30] define *trust* como: '*confidence in or reliance on some quality or attribute of a person or thing, or the truth of a statement*', ou seja, convicção ou confiança em alguma qualidade ou atributo de uma pessoa ou coisa, ou a verdade de uma declaração. Em [38], Cysneiros e Werneck apresentam exemplos que sugerem

que *trust* está intimamente relacionada a questões de transparência e citam ainda que a transparência é essencial para garantir *trust*, ou seja, há uma relação de potencialização bilateral entre os dois aspectos.

Entendemos que a correlação entre *provenance* e *trust* pode influenciar positivamente em questões de transparência. Assim, nosso objetivo no trabalho foi a utilização da aplicação de conceitos de *provenance* para que a interação dos agentes MONITOR e ANALISADOR fosse baseada em objetivos, planos e rastros de execução do agente ANALISADOR. Dessa forma, procuramos dar uma solução heurística, que consiste em efetuar uma monitoração a partir de uma base de conhecimentos de monitorações prévias de transparência com objetivo em prover maior confiança nos dados e agilidade no processo de análise e geração dos resultados. Além disso, a estratégia de *provenance* permite o acumulo de conhecimento a partir de diversas monitorações de *software* que formam uma base de dados dos modelos canônicos utilizados e as monitorações que registraram conformidades com catálogo de transparência. A partir dessa abordagem, a proposta de monitoração pretende estar aderente a um conceito mais amplo de *provenance*, que está relacionado não somente na origem da informação, mas também de como ela é derivada, o seu contexto e como é utilizada.

Para prover traços de execução de *provenance* o agente ANALISADOR mantém um rastro de sua execução no momento da comparação das *tags* dos modelos canônicos do *software* monitorado com as alternativas, questões e grupos do catálogo de transparência e esse rastro passa a servir como heurística para novas monitorações formando uma base de conhecimento de conformidades de transparência.

Na proporção em que diversos *software* forem monitorados, há o crescimento da base de conhecimento para dar suporte à heurística de monitorações a fim de minimizar a necessidade do uso de modelos canônicos. A consequência seria uma monitoração mais ágil evitando a interação dos agentes do sistema de monitoração.

A Figura 6 apresenta a visão macro do processo de monitoração. A análise deve ser feita considerando a figura como um todo, o que representa o processo de monitoração a partir do modelo canônico. Nesse caso, deve ser desconsiderada a interação entre o agente MONITOR e o ANALISADOR a partir do recurso dos traços de execução de *provenance*. Pode-se verificar nessa representação que a monitoração requer a interação dos diversos agentes do sistema.

O recorte superior da figura já representa uma otimização do processo de monitoração. A interação entre os agentes MONITOR e ANALISADOR requer apenas os traços de execução do *software* monitorado e os de *provenance* para que seja avaliada e sugerida transparência do *software*, uma vez que o agente MONITOR utiliza os traços de execução de *provenance* como heurística de monitoração e transfere o processamento para o ANALISADOR ao invés de caminhar com o processo para uma varredura mais completa a partir dos agentes CANONIZADOR E CONSOLIDADOR.

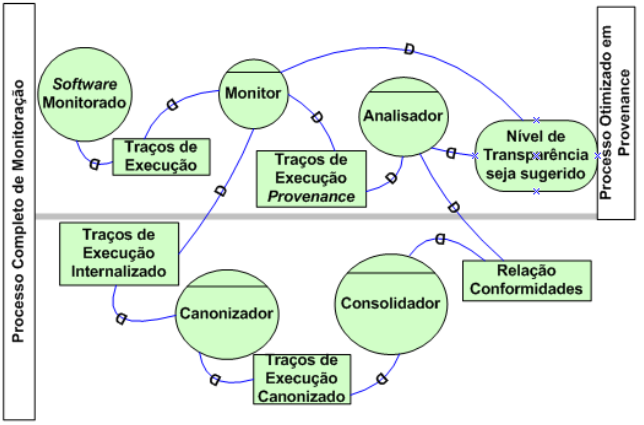

Figura 6. Visão macro do processo de monitoração – *framework i** [19].

O fluxo baseado em registros de *provenance* evita o uso de todos os agentes no processo de monitoração. Essa estratégia é baseada na disponibilidade de uma base de conhecimento histórica para dar suporte às decisões do agente MONITOR. Essa base de conhecimento, armazenada a partir de uma abordagem de *provenance*, permite ao MONITOR ter acesso às informações de monitorações anteriores e decidir pela sugestão direta ao agente ANALISADOR que o traço de execução monitorado no momento já possui uma correlação de semelhança com traços de execuções já monitorados.

O MONITOR verifica nos registros de *provenance* os itens do 5w2h sugeridos no modelo canônico e armazenados com suas correlações com itens do catálogo de transparência. Dessa forma, já há uma predisposição para que o MONITOR direcione diretamente o traço de execução monitorado no momento para que o agente ANALISADOR consolide e exiba os resultados da monitoração e a indicação de transparência.

Os registros de *provenance* são armazenados em formato XML a partir de cada avaliação do agente ANALISADOR. Os conteúdos são registrados entre as *tags* propostas conforme modelo canônico (Tabela 3), ou seja, à medida que o ANALISADOR encontra uma evidência baseada em tal modelo, ele grava o conteúdo encontrado entre as *tags* indicadoras da proposta do 5w2h. Por exemplo, ao pesquisar sobre o atributo Rastreabilidade de transparência, o agente registra conteúdo do item analisado entre as *tags* *<what> ... </what>*.

O exemplo apresenta parte da estrutura armazenada após a análise do LS. É possível verificar os conteúdos armazenados entre as *tags* 5w2h e também alguns conteúdos em branco que indicam que não houve uma evidência.

```
1.  <!-- provenance record structure -->
2.  <expressionRatio>
3.     <uniqueID>PROVENANCE-00001-LattesScholar</uniqueID>
```

```
4.    <what>Rastreabilidade</what>
5.    <why>Transparência de Software</why>
6.    <where>//SERVIDOR/C/PASTA/managerAgent-2010-09-1315-43-
      24.txt</where>
7.    <provenanceRecord>
8.       <idProvRec>1</idProvRec>
9.       <whatProv>Data de registro</whatProv>
10.      <whatContent>20100913</whatContent>
11.      <whenProv>20100913 15:43:24</whenProv>
12.      <where>posição 1 a 8</whatContent>
13.      <howProv>AAAAMMDD</howProv>
14.      <howMuchProv></howMuchProv>
15.      <transparencyCatalogue>
16.        <alternativePattern>1.1.2</alternativePattern>
17.        <situation>atende</situation>
18.      <transparencyCatalogue>
19.    </provenanceRecord>
20.
21.    <provenanceRecord>
22.      <idProvRec>2</idProvRec>
23.      <whatProv>Hora de registro</whatProv>
24.      <whatContent>15:43:24</whatContent>
25.      <whenProv>20100913 15:43:24</whenProv>
26.      <where>posição 9 a 18</whatContent>
27.      <howProv>HH:MM:SS</howProv>
28.      <howMuchProv></howMuchProv>
29.      <transparencyCatalogue>
30.        <alternativePattern>1.1.2</alternativePattern>
31.        <situation>atende</situation>
32.      <transparencyCatalogue>
33.    </provenanceRecord>
34.
35.    <provenanceRecord>
36.      <idProvRec>3</idProvRec>
37.      <whatProv>Nome do agente</whatProv>
38.      <whatContent>Agente 1</whatContent>
39.      <whenProv>20100913 15:45:28</whenProv>
40.      <where>posição 1 a 8</whatContent>
41.      <howProv>Agente name=</howProv>
42.      <howMuchProv></howMuchProv>
43.      <transparencyCatalogue>
44.        <alternativePattern>1.1.1</alternativePattern>
45.        <situation>atende</situation>
46.      <transparencyCatalogue>
47.    </provenanceRecord>
48.    <...>
```

```
49.  </expressionRatio>
```

5 Mapeamento do modelo intencional para BDI e Jadex

A implementação dos agentes monitores de transparência foi baseada em um mapeamento entre o modelo intencional, a arquitetura BDI e a codificação em Jadex [32], conforme proposto em [29]. A Figura 7 apresenta a proposta do mapeamento entre as diferentes camadas.

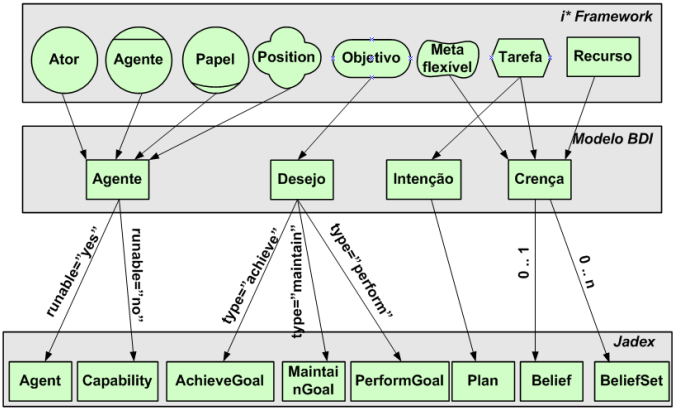

Figura 7. Mapeamento i* x BDI x Jadex [29].

A descrição textual da figura propõe a estratégia de evolução do processo de implementação dos agentes. Os agentes não executáveis originados de papéis ou posições deveriam ser implementados como capacidades; os desejos foram traduzidos como objetivos e desenvolvidos ou mantidos de acordo com a *tag type* do modelo BDI; as intenções foram implementadas com planos, ou seja, classes Java que estenderam os planos da plataforma Jadex [32]; as crenças com cardinalidades de 0 para 1 ou 1 para 1 foram traduzidas para *beliefs* Jadex, enquanto que crenças com cardinalidades de 0 para n e 1 para n foram traduzidas como conjuntos de *beliefset* Jadex, conforme proposto em [29]. Em nossa proposta de implementação o catálogo de transparência foi descrito nas crenças do agente ANALISADOR a partir dos *beliefs*, que passaram a indicar as questões dos *patterns* de transparência e os *beliefset* que tiveram o objetivo para indicar as alternativas.

A especificação BDI dos agentes foi feita em uma arquitetura baseada em XML. O trecho de código da página subsequente apresenta um arcabouço da estrutura do agente ANALISADOR baseado na proposta de [29] do mapeamento i* para BDI e Jadex. Alguns trechos de código foram suprimidos devido à extensão do código fonte, mas a estrutura procura refletir a arquitetura utilizada na implementação.

A implementação visa a refletir as características da política de monitoração apresentada no modelo intencional da Figura 5. Apesar de funcional, ela ainda é um esforço inicial de codificação para operacionalizar as questões de monitoração, tal código fonte poderá ser revisto em outro momento para atender a melhores condições de arquitetura e qualidade.

O código está subdivido na parte XML que caracteriza os agentes e no código Java com a representação de classes elaboradas para atender aos planos.

```
1.  <!-- Agente ANALISADOR - Representação XML -->
2.  <agent Name="AnalyzerAgent" package="analyzerAgent">
3.    <imports> ... </imports>
4.    <beliefs>
5.       <belief name="traceabilityPatterns"
6.               class="String">
7.         <fact>c:\..\traceabilityPatterns.xml</fact>
8.       </belief>
9.
10.      <belief name="canonicalModel"
11.              class="String">
12.        <fact>c:\..\canonicalModel.xml</fact>
13.      </belief>
14.
15.      <belief name="canonicalTrace"
16.              class="String">
17.        <fact>c:\..\canonicalTrace.xml</fact>
18.      </belief>
19.      <...>
20.   </beliefs>
21.
22.   <...>
23.
24.   <!-- Plan for receive the execution trace. -->
25.   <plan name="receiveExecTracePlan">
26.      <body class="ReceiveExecTrace"/>
27.   </plan>
28.   <plans>
29.   <!-- Plan for receive the traceability patterns -->
30.   <plan name="receiveTraceabilityPatternsPlan">
31.      <body class="receiveTraceabilityPatterns "/>
32.   </plan>
33.    <!-- Plan for evaluate the transparency according
34.         the operacionalizations. -->
35.    <plan name="evaluateTransparencyExecTrace">
36.       <body class="EvaluateTransparencyPlan"/>
37.       <trigger>
38.          <internalevent ref="call_Evaluate TransparencyPlan"/>
39.       </trigger>
```

```
40.     </plan>
41.
42.     <!-- Plan for provenance registration. -->
43.      <plan name="provenanceRegister">
44.        <body class="provenanceRegisterPlan"/>
45.         <trigger>
46.            <internalevent ref="call_provenanceRegisterPlan"/>
47.         </trigger>
48.      </plan>
49.
50.      <...>
51.
52. </plans>
```

```
1.  <!-- Agente ANALISADOR - Classes Java -->
2.  public class traceabilityPatterns extends Plan {
3.     <...>
4.     public void body() {
5.       patternsDefinitions traceabilityPatterns =
6.       (patternsDefinitions)getBeliefbase().
7.       getBelief("traceabilityPatterns").getFact();
8.      <...>
9.  <...>
10. public class canonicalTrace extends Plan {
11.    <...>
12.    public void body() {
13.      canonicalTraceDefinitions canonicalTrace =
14.      (canonicalTraceDefinitions)getBeliefbase().
15.      getBelief("canonicalTrace").getFact();
16.     <...>
17.
18.     OpsOftraceability traceabilityEvaluation = new
          OpsOftraceability();
19.
20.     <...>
```

O trecho de código inicial procura demonstrar as partes bem definidas em padrão XML com as seções de *beliefs* e os planos do agente, por exemplo, a base de conhecimento do agente ANALISADOR está caracterizada sob os `beliefs traceabilityPatterns`, que possui informações dos *patterns* de transparência com a descrição das questões e alternativas de rastreabilidade; canonicalModel possui o modelo canônico formatado em padrão específico para ser utilizados pelos planos nas comparações e validações de conformidades dos *patterns* de transparência; canonicalTrace possui um modelo canonizado previamente pelo agente CANONIZADOR para simplificar o processo de comparação e identificação de conformidades.

A segunda parte do código representa a configuração dos planos dos agentes e suas classes Java, essas últimas basicamente cuidam da captura dos *beliefs* do agente.

Outro código, que será apresentado posteriormente representa as comparações de verificação de conformidades e não conformidades nos registros dos traços de execução canonizados para que sejam validadas as alternativas dos *patterns* de rastreabilidade. O retorno dessas comparações será utilizado para a exibição dos resultados de conformidades, ou não, de acordo com as colunas apresentadas na Tabela 4 identificadas com a letra A.

A comparação é feita a partir das informações do modelo canônico e as questões de transparência. O modelo canônico foi desenvolvido com o objetivo de registrar as informações dos traços de execução em um único formato generalizado. Para isso, ele foi projetado conforme uma estrutura de processo, composto por atividades que, por sua vez, são compostas por atores e informações.

Existem diferentes tipos de traços de execução, por exemplo, um traço de execução de atividades sobre um banco de dados. Nesse caso, possivelmente terá registros de ações (como inserir e apagar dados) as quais serão realizadas por alguém (ator), em uma determinada data e hora. Neste caso, o traço de execução não seguirá um processo definido, entretanto, um sistema de workflow poderá criar traços de execução sequenciais onde serão registrados detalhes de um dado processo em execução, o que possibilita explorar suas características de registro temporal (execução cronológica de atividades) e registro de transformação das informações envolvidas. Portanto, independente do conjunto de informações presentes nos traços de execução, entende-se que os registros sempre poderão preencher o modelo canônico, mesmo que parcialmente, uma vez que a natureza do traço de execução é registrar, de uma forma geral, sentenças compostas por ações (*activity*) executadas por algum ator (*actor*) em um dado momento.

A representação do modelo pode ser vista conforme a Figura 8. Os atores representam os registros dos responsáveis pelas ações nos traços de execução que, por sua vez, são representados pelas atividades; as informações são possíveis registros de entrada e saída do conteúdo das atividades; o processo registra o conjunto de atividades que são extraídas de informações como data e hora de inicio e fim de sua execução. Esses registros somente serão úteis se os traços de execução estiverem seguindo algum processo pré-definido.

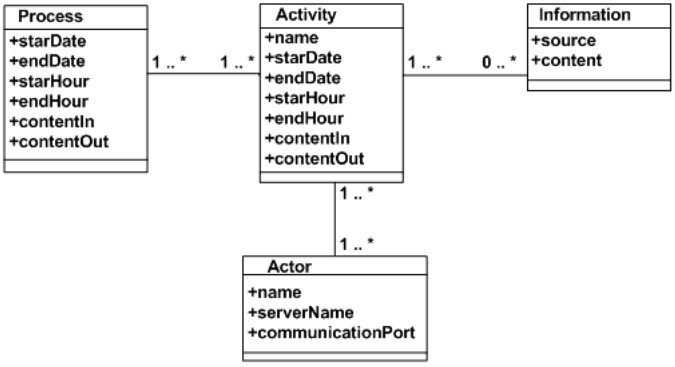

Figura 8. Representação do modelo canônico.

Após preencher o modelo canônico com as informações dos traços de execução, a avaliação da transparência, neste caso a rastreabilidade, é executada por funções que verificam se as informações esperadas para as alternativas dos *patterns* estão presentes nos respectivos campos do modelo canônico. Portanto, há uma relação entre as alternativas e cada campo do modelo canônico. O código exemplifica a verificação da alternativa 1.1.1.

```
1.  Public OpsOftraceability VerifyOp111 (
2.          traceabilityDefinition traceability,
3.          OpsOftraceability traceabilityEvaluation)
4.  {
5.      if((traceability.getProcess().getContentIn() != null) &&
6.      (traceability.getProcess().getContentOut()!= null)     &&
7.      traceability.getProcess().getActivity(0).getContentIn()!=
    null) &&
8.      (traceability.getProcess().getActivity(0).getContentOut()!=
    null))
9.      {
10.        traceabilityEvaluation.setOp111(true);
11.     else
12.        traceabilityEvaluation.setOp111(false);
13.     }
14.     return traceabilityEvaluation;
15. }
```

Já o trecho de código demonstra a comparação das alternativas dos *patterns* de rastreabilidade para que o agente possa definir como verdadeira a conformidade da questão, identificadas na Tabela 4 com a letra Q. Por exemplo, se a alternativa representada por `traceabilityEvaluation.isOp111()` for indicada como *true* e a `traceabilityEvaluation.isOp112()` também for, a questão 1.1 estaria em

conformidade com o catálogo de transparência para rastreabilidade, do contrário há uma violação da regra.

```
1.  public OpsOftraceability VerifyOp11(
2.          traceabilityDefinition traceability,
3.          OpsOftraceability traceabilityEvaluation)
4.          {
5.          if (traceabilityEvaluation.isOp111() &&
6.              traceabilityEvaluation.isOp112())
7.              traceabilityEvaluation.setOp11(true);
8.          else
9.              traceabilityEvaluation.setOp11(false);
10.         return traceabilityEvaluation;
11.         }
```

5.1 Resultados obtidos

A Tabela 4 apresenta o relatório exibido ao final da monitoração. Está presente na tabela um recorte dos resultados obtidos da monitoração da transparência do LS. É possível perceber que a coluna *Atende* indica que vários elementos da rastreabilidade estão em conformidade com os *patterns* de transparência, mas há alguns elementos que violam a regra. A indicação de *Atende* ou *Violação de regra* está evidenciada com um X. O X é marcado nas opções dos *patterns* (T, Q ou A) a partir do retorno das classes exemplificadas na seção anterior, principalmente pelo retorno da classe `OpsOftraceability` a partir de traceabilityEvaluation.

É possível verificar no relatório da Tabela 4 que várias alternativas estão de acordo com o esperado nas respostas às perguntas do *patterns* de transparência. A positivação dessas alternativas influencia positivamente sua questão, que por consequência, valida o grupo.

Entendemos que o *software* monitorado viola algumas respostas que não são atendidas nos *patterns* de transparência. No resultado listado na Tabela 4, nota-se que a alternativa 1.2.1 não possui uma correlação positiva no modelo canônico. A consequência é uma indicação de que o *software* viola a regra definida para a questão. O relatório final é apresentado de forma parcial para exemplificar o resultado da monitoração.

O relatório com os resultados obtidos poderá servir como subsídio para tomada de decisão dos desenvolvedores do *software*. Assim, esses poderão atuar de forma a resolver as questões e alternativas que não estão em conformidade com o esperado de transparência, caso queiram tornar seu *software* mais transparente.

Tabela 4. Resultados obtidos.

Rastreabilidade: Grupos / Questões / Alternativas	Clas	Atende	Violação de Regra
1. Fazer Pré-Rastreabilidade.	G	X	
1.1 As fontes de informação utilizadas são rastreadas?	Q	X	
1.1.1 São identificados atores que passam a informação.	A	X	
1.1.2 É identificado o momento em que a informação foi coleta.	A	X	
1.2 Os recursos utilizados ou que serão necessários são rastreados?	Q		X
1.2.1 Identificação de erros (*error*...), cabeçalhos de traços de execução(resumo, objetivo, justificativa, número de linhas geradas).	A		X
1.3 As interações entre interessados são rastreadas?	Q	X	
1.3.1 registros de troca de mensagens.	A	X	
...	...	...	...
(continua)	...	...	...

6 Conclusão

O presente artigo apresentou um primeiro ensaio de monitoração para TS. A proposta foi baseada em um mapeamento entre os modelos intencional do *framework* i*, o BDI e a plataforma Jadex para SMA.

O foco inicial da monitoração foi a verificação dos traços de execução do LS e a verificação da conformidade dos registros em relação aos *patterns* de transparência. Os registros de não conformidades estão apenas registrados no sistema e serão explorados em trabalhos futuros com a proposta do uso de sistemas de recomendação [34] [36]. A intenção é criar agentes autônomos inteligentes que possam sugerir alternativas a partir do catalogo de transparência para que as não conformidades encontradas sejam mitigadas.

O trabalho assemelha-se com o descrito em [35], que apresenta uma arquitetura chamada COMMAS (*COndition Monitoring Multi-Agent System*) desenvolvida para monitorar e interpretar dados a partir da detecção e distinção de falhas no *software* monitorado. Para Mangina, McArthur e McDonald [35] as falhas encontradas precocemente pela monitoração podem trazer benefícios na redução de custos operacionais, na melhoria da economia de instalações em uma fábrica e na melhoria de segurança. O trabalho tem convergência no ponto de utilização de bases de conhecimento e em modelos baseados em raciocínio. Há também uma relação com o trabalho relatado em [27], que dão um tratamento de registro da documentação de um processo de doação de órgãos. O sistema proposto em [27] documenta de forma explicita os objetivos dos agentes e suas conexões com o processo a partir de um modelo de interação entre

agentes autônomos e não-autônomos. Para isso adotam a estratégia do uso de SMA e *provenance*.

A proposta do modelo intencional para políticas de monitoração de transparência baseada em SMA está sendo evoluída, bem como a implementação dos agentes autônomos e não se esgota com o exposto até aqui. A monitoração do LS foi uma primeira operacionalização prática de monitoração para análise de transparência e deve ser melhor explorado para aumentar sua confiança. Entendemos que é necessária uma monitoração mais sistemática e a sugestão de novos modelos canônicos que ofereçam suporte para a monitoração de outros GT do catálogo com o objetivo de explorar melhor as variabilidades dos *patterns* de cada um desses graus. O catálogo é algo que também está em fase de evolução e melhoria contínua, novas alternativas poderão surgir a medida haja uma reavaliação e validação por parte do Grupo ER - PUC-Rio.

Referências

1. Fung, A.; Graham, M.; Weil, D. (2007). Full Disclosure, the Perils and Promise of Transparency, Cambridge University Press.
2. Holzner, B.; Holzner, L. (2006). Transparency in Global Change: The Vanguard of the Open Society. University of Pittsburgh Press; 1 edition.
3. Lord, K. M. (2006). The Perils and Promise of Global Transparency, State University of New York Press.
4. Leite, J. C. S. do P.. Sistemas de Software Transparentes. Palestra convidada do 20 Simpósio Brasileiro de Engenharia de Software. Outubro de 2006. <Disponível em: http://www.di.inf.puc-rio.br/~julio/slct-pub/transp-sbes.pdf> <Acessado em: 22/12/2012>
5. Grupo de Pesquisas em Engenharia de Requisitos da PUC-RIO. <Disponível em: http://www.er.les.inf.puc-rio.br/~wiki/index.php > <Acessado em: 20/12/2012>
6. Transparência de Software. <Disponível em: http://www.er.les.inf.puc-rio.br/~wiki/index.php/Transpar%C3%AAncia_de_Software> <Acessado em: 20/12/2012>
7. Wooldridge, M. (2002). *An Introduction to Multi-Agent Systems*. John Wiley and Sons.
8. Jennings, N. R.; Wooldridge, M.. (2002) Agent-Oriented Software Engineering in Handbook of Agent Technology (ed. J. Bradshaw) AAAI/MIT Press.
9. Cappelli, C.; Leite, J. C. S. P. (2008). Transparência de Processos Organizacionais. Universidade Federal Fluminense, LATEC. II Simpósio Internacional de Transparência nos Negócios.
10. Cappelli, C. (2009). Uma Abordagem para Transparência em Processos Organizacionais Utilizando Aspectos. Rio de Janeiro. 328 p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.
11. Chung, L.; Nixon, B.; YU, E.; Mylopoulos, J. (2000). Non-Functional Requirements in Software Engineering – Kluwer Academic Publishers – Massachusetts, USA.
12. Catálogo de Transparência de Software. <Disponível em: http://www.er.les.inf.puc-rio.br/~wiki/index.php/Cat%C3%A1logo_Transpar%C3%AAncia>
    <Acessado em: 20/12/2012>
13. Lattes-Scholar: Requirements Engineering Group at PUC-Rio. <Disponível em:

http://www.er.les.inf.puc-rio.br/~wiki/index.php/Lattesscholar> <Acessado em: 20/12/2012>

14. Currículo Lattes. <Disponível em: lattes.cnpq.br > <Acessado em: 21/12/2012>
15. Google Scholar. <Disponível em: www.scholar.google.com > <Acessado em: 21/12/2012>
16. Publish and Perish. <Disponível em: http://www.harzing.com/pop.htm> <Acessado em: 22/12/2012>
17. Microsoft Academic Search. <Disponível em: http://academic.research.microsoft.com> <Acessado em: 22/12/2012>
18. Google Scholar Citation <Disponível em: http://scholar.google.com/intl/en/scholar/citations.html> <Acessado em: 22/12/2012>
19. Yu E.S.K. (1995): Modelling Strategic Relationships For Process Reengineering. Ph.D. dissertation. Dept. of Computer Science, University of Toronto.
20. Oliveira, A. P.; Leite, J. C. S. P.; Cysneiros, L. M. (2008). Método ERi*c - Engenharia de Requisitos Intencional. In: 11th Workshop on Requirements Engineering, Barcelona. Proceedings of the 11th Workshop on Requirements Engineering. Barcelona: Universitat Politécnica de Catalunya. p. 155-166.
21. Rao, A.S.; Georgeff, M.P. (1992). An abstract architecture for rational agents. In: International Conference on Principles of Knowledge Representation and Reasoning - KR, 3., 1992. Proceedings. Morgan Kaufman.
22. Bratman, M. (1987). Intention, Plans, and Practical Reason. Harvard University Press. Cambridge, MA, USA.
23. Bratman, M.; Israel, D.; Pollack, M. (1987). Toward an architecture for resource-bounded agents. Stanford: Stanford University.
24. Bratman, M. (1984). Two faces of intention. The Philosophical Review, v.93, n.3, p.275-405.
25. Bratman, M. (1989). What is intention? In: COHEN, P; MORGAN, J; POLLACK, M. (Eds.), 1989. Anais. MIT Press.
26. Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language, in 'MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world: agents breaking away', Springer-Verlag New York, Inc., Secaucus, NJ, USA, pp. 42–55.
27. Miles, S.; Munroe, S.; Luck, M.; Moreau, L. (2007). Modelling the provenance of data in autonomous systems', in Proceedings of Autonomous Agents and Multi-Agent Systems 2007, pp. 243–250, Honolulu, Hawai'i, May.
28. Miles, S.; Groth, P.; Branco, M.; Moreau, L. (2007). The requirements of using provenance in e-science experiments, Journal of Grid Computing, 5, 1–25.
29. Serrano, M.; Leite, J. C. S. P. (2011). Development of Agent-Driven Systems: from i* Architectural Models to Intentional Agents Code. In: Fifth International istar Meeting, 2011, Itália. Fifth International istar Meeting.
30. Oxford English Dictionary, "provenance, n". Oxford University Press.
31. Pinheiro da S.; P. McGuinness; D. McCool. (2003). Knowledge Provenance Infrastructure. IEEE Data Engineering Bulletin 26(4), pp. 26-32.
32. Braubach, L.; Lamersdorf, W.; Pokahr, A. (2003). Jadex: Implementing a BDI Infrastructure for

JADE Agents. Distributed Systems and Information Systems, vol. 3, n. 3, pp.76-85, September.

33. Vasquez I.; Gomadam K.; Patterson S. (2005). Framework for representing provenance for web services and processes. Technical Report, LSDIS Lab.
34. Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction, Vol. 12(4), 331-370.
35. Mangina, E.; and McArthur, S.D.J; and McDonald, J.R. (2001). COMMAS (COndition Monitoring Multi-Agent System). Autonomous Agents and Multi-Agent Systems, 4 (3). pp. 279-282. ISSN 1387-2532
36. Adomavicius, G.; Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, Vol. 17(6), 734-749.
37. Mayer, R. C.; Davis, J. H.; and Schoorman F. D. (1995). An integrative model of organizacional trust. The Academy of Management Review, 1995. 20 (3): p. 709-734.
38. Cysneiros, L. M.; Werneck, V. M. B. (2009). An Initial Analysis on How Software Transparency and Trust Influence each other. In: 12th Workshop on Requirements Engineering, Proceedings of, 2009, Valparaiso. Proceedings of 12th Workshop on Requirements Engineering,. Valparaiso: Universidad Tecnica Federico Santa Maria, 2009.v.1.p. 27-32.
39. Serrano, M.; Sampaio do Prado Leite, J.C. (2011) Capturing transparency-related requirements patterns through argumentation. In: First International Workshop on Requirements Patterns (RePa), pp.32-41, 29 Aug. 2011.
40. Basili, V.R. (1992) Software Modeling and Measurement: The Goal Question Metric Paradigm. Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, September 1992.