Transformação do Modelo i* em User Stories: uma abordagem para documentação ágil baseada nas razões, intenções e NFR

Celso Agra[1], Denise Assis[2], Aêda Sousa[3], Aline Jaqueira[4], Márcia Lucena[4], Teresa Maciel[5] e Fernanda Alencar[3]

[1] Centro de Estudos e Sistemas Avançados do Recife, Recife, Brasil
[2] Universidade de Pernambuco, Recife, Brasil
[3] Universidade Federal de Pernambuco, Recife, Brasil
[4] Universidade Federal do Rio Grande do Norte, Natal, Brasil
[5] Universidade Federal Rural de Pernambuco, Recife, Brasil

clasf@cesar.org.br, dabs@ecomp.poli.br, {aedasousa, alineopj, marcia.lucena, tmmaciel, fernandaalenc}@gmail.com

Abstract. Na documentação ágil, razões, intenções e requisitos não funcionais nem sempre são considerados a contento durante o ciclo de vida de desenvolvimento de um produto. A não consideração desses aspectos pode prejudicar a qualidade do sistema, levando a dificuldades na manutenção e no gerenciamento dos requisitos. Para mitigar o problema, este trabalho propõe a transformação do modelo i* (iStar), que naturalmente considera razões, intenções e requisitos não funcionais desde as fases iniciais, através das *user stories*. A transformação é realizada através de um conjunto de diretrizes que inclui o *design rationales* na documentação ágil. Para análise da proposta, considerou-se os exemplos propostos por Eric Yu na concepção do i*. Como resultado, é apresentada uma documentação que considera razões, intenções e requisitos não funcionais, evitando assim a construção de requisitos desnecessários e minimizando o impacto das mudanças de requisitos.

Keywords. métodos ágeis; modelo de transformação; i*; *user stories*; razões; intenções; requisitos não funcionais; *design rationales*; engenharia de requisitos orientada a objetivos.

1 Introdução

O desenvolvimento de *software* ágil está sendo aplicado cada vez mais na indústria e também se tornando um assunto popular no meio acadêmico [9]. Isso significa que a adoção de práticas ágeis é uma alternativa realmente viável aos modelos tradicionais existentes. Os métodos ágeis contribuem para o desenvolvimento de software, fornecendo uma visão clara do sistema mesmo em ambientes que apresentam constantes mudanças. A metodologia ágil permite desenvolver sistemas que terão seus requisitos alterados ou que ainda não estão bem definidos, além de limitar-se a documentar apenas o essencial para a manutenção e construção do software. Entretanto, existem al-

guns desafios, que serão abordados a seguir, e que requerem uma atenção na construção de requisitos para sistemas apoiados pelos métodos ágeis. A documentação ágil está sendo bastante questionada, pois ainda não há um consenso sobre a melhor forma de documentar e compartilhar informações através de um meio eficiente.

Para Hoda, Noble e Marshall [5], os métodos ágeis não possuem uma definição clara do que é uma documentação necessária para o produto de software e, portanto, a falta dessa compreensão causa problemas para definir o necessário para o produto final. Em [13], é afirmado que uma vez realizada a documentação, é preciso mantê-la e atualizá-la sempre que houver mudanças de requisitos, o que gera um custo extra durante o desenvolvimento de software. De acordo com [15], a comunicação verbal é suscetível a perda de informações durante o ciclo de vida do produto, pois em muitos casos, as razões e intenções do sistema não são lembradas, nem documentadas pela equipe. Por último, [9] afirma que os problemas apresentados em projetos, documentação e código, estão normalmente relacionados com a negligência da equipe, pois não possui conhecimento em produzir uma documentação apropriada para o projeto. Logo, a documentação ágil ainda apresenta lacunas a serem preenchidas, pois os documentos gerados nem sempre atendem as necessidades do projeto, e em alguns casos, não possuem uma visão clara e explícita das razões, intenções, arquitetura, requisitos funcionais e não-funcionais, para o sistema proposto.

É comum que projetos ágeis não utilizem uma documentação vasta e nem uma modelagem abrangente para representar requisitos e as necessidades do software. Isso pode dificultar a análise de informações complexas e na compreensão dos conflitos da solução, além de fornecer uma base limitada para avaliar o impacto das mudanças dos requisitos [10]. Assim, é preciso ter uma visão geral do sistema antes da fase de projeto e implementação, focando na fase de análise de requisitos, e identificando os objetivos do sistema proposto. A documentação ágil possui lacunas que precisam ser exploradas para garantir a completude das informações, sem ferir, no entanto, os valores do manifesto ágil [1]. De acordo com [13], uma documentação que abrange um alto nível abstração dos requisitos, fornece as razões e as justificativas para as decisões das funcionalidades e, portanto, permite que os *stakeholders* compreendam as intenções de cada ação do sistema. Os documentos que expressam as razões e justificativas para as decisões do sistema são conhecidos como *design rationales*.

Para [8], os documentos de *design rationale* precisam ser representados na forma de modelos e estão associados com a *Goal-Oriented Requirement Engineering* (GORE), que define os objetivos de forma clara e compreensível para os *stakeholders* [14]. A modelagem GORE apresenta metas organizacionais e os porquês das necessidades de um determinado sistema [2], ou seja, definem as razões e as intenções. Alguns modelos ainda possuem a representação de requisitos não funcionais [8].

Este trabalho apresenta uma proposta de transformação do modelo i* em *user stories* para suprir as necessidades encontradas na documentação ágil, como a construção de uma documentação baseada nas razões, intenções e requisitos não funcionais. O uso de uma abordagem GORE como um documento de *design rationale* contribui para evitar que requisitos desnecessários possam ser desenvolvidos, além de mitigar o impacto das mudanças desses requisitos. Escolhemos o i* por ser uma abordagem GORE, que possui uma modelagem bem consolidada com apoio de ferramentas ade-

quadas [2]. Além disso, o i* é adequado para representar os objetivos organizacionais e as possíveis interações entre os atores, utilizando conceitos como metas e metas-flexíveis (*softgoals*) [12].

2 Framework i*

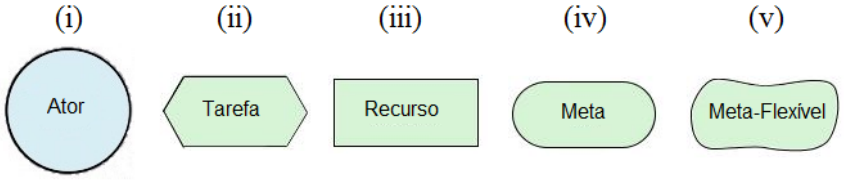
O *framework* captura os requisitos intencionais utilizando relacionamentos estratégicos entre os atores. Assim, o *framework* fornece um entendimento das razões ou dos "porquês" que se aplicam aos requisitos do sistema [11]. O *framework* está dividido em dois modelos: o *Strategic Dependency* (modelo SD ou modelo estratégico de dependências); e o *Strategic Rationale* (modelo SR ou o modelo estratégico de razões).

2.1 Elementos do i*

O *framework* i* possui elementos utilizados em ambos modelos (SD e SR). Assim, alguns conceitos relacionados aos elementos foram definidos da seguinte forma:

- Ator: é representado pelo elemento (i) da Figura 1. Para o modelo i*, o ator possui intenções (visão externa) e capacidades (visão interna). Além disso, pode se relacionar com outros atores através das relações de dependências.
- Tarefa: realiza tarefas e operações de um determinado ator ou relacionamento, que serão realizadas. Uma tarefa, representada pelo elemento (ii) da Figura 1.
- Recurso: pode ser caracterizado como um recurso físico que pode ser obtido por um ator. Um recurso está representado pelo elemento (iii) da Figura 1, e pode caracterizar um documento, por exemplo.
- Meta: representa os objetivos intencionais de um ator. Uma meta está representada pelo elemento (iv) da Figura 1.
- Meta-flexivel: representa as metas relacionadas a qualidade ou objetivos secundários. Uma meta está representada pelo elemento (v) da Figura 1 e pode caracterizar um requisito não funcional.

Figura 1: Representação dos elementos do i*



2.2 Modelo SD

Este modelo está focado nos relacionamentos intencionais entre os atores, pois cada ligação representa uma dependência entre eles, e é representado por nós e ligações (ou conectores). Os nós são representados pelos atores, que se dividem em dois grupos: o *depender*, um ator dependente de um determinado elemento (denominado *dependum*); e o *dependee*, que representa o ator que fornece o elemento para o *depender*. As liga-

ções são as dependências entre os atores, e são representadas por elementos intencionais (*Intentional Elements*), como as metas (*goals*), as metas flexíveis (*softgoals*), os recursos (*resources*) e as tarefas (*tasks*).

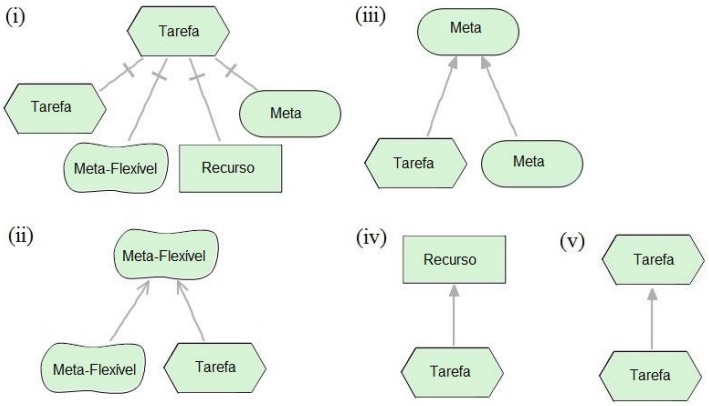
A Figura 2 apresenta uma representação de um modelo SD. Nesse caso, teremos uma ligação entre os atores *depender* e *dependee* através do *dependum*. Nesse caso, o depender necessita do *dependee* para realizar ou executar o *dependum*.

Figura 2: Análise realizada em um modelo SD apresentando os atores *depender* e *dependee*, e o elemento intencional do *dependum*

2.3 Modelo SR

O modelo SR expande a descrição de um determinado ator e mostra as suas razões associadas e suas dependências. Esse modelo apresenta todas as justificativas envolvidas em suas interações, e fornece um apoio para modelar as razões de cada ator em seus relacionamentos intencionais [2]. Nesse caso, o modelo SR representa os elementos internos do modelo i*. O *framework* possui diversas variações provenientes da evolução do modelo devido a necessidade de realizar uma melhoria desse *framework*. Para a proposta deste trabalho, resolvemos utilizar o modelo tradicional SR, baseado no trabalho de [16]. A Figura 3 mostra como essas ligações são utilizadas.

- Task-decomposition: representado pelo ítem (i) da Figura 3. Uma tarefa pode ser decomposta em partes, que podem representar qualquer um dos quatro tipos de elementos, como meta-flexível, tarefa, recurso e meta.
- Contribution: representado pelo item (ii) da Figura 3. Está relacionado com a contribuição de uma determinada meta-flexível (*softgoal*) ou requisito não funcional.
- Means-Ends: representado pelos itens (iii), (iv) e (v) da Figura 3. O relacionamento entre o meio (*means*), que pode ser uma tarefa ou uma meta a ser realizada, e o fim (*end*), que pode ser um objetivo, uma tarefa ou um recurso a ser alcançado, de acordo com o elemento fim.
    - **Figura 3:** Representação do modelo SR do i* tradicional baseado em [16]

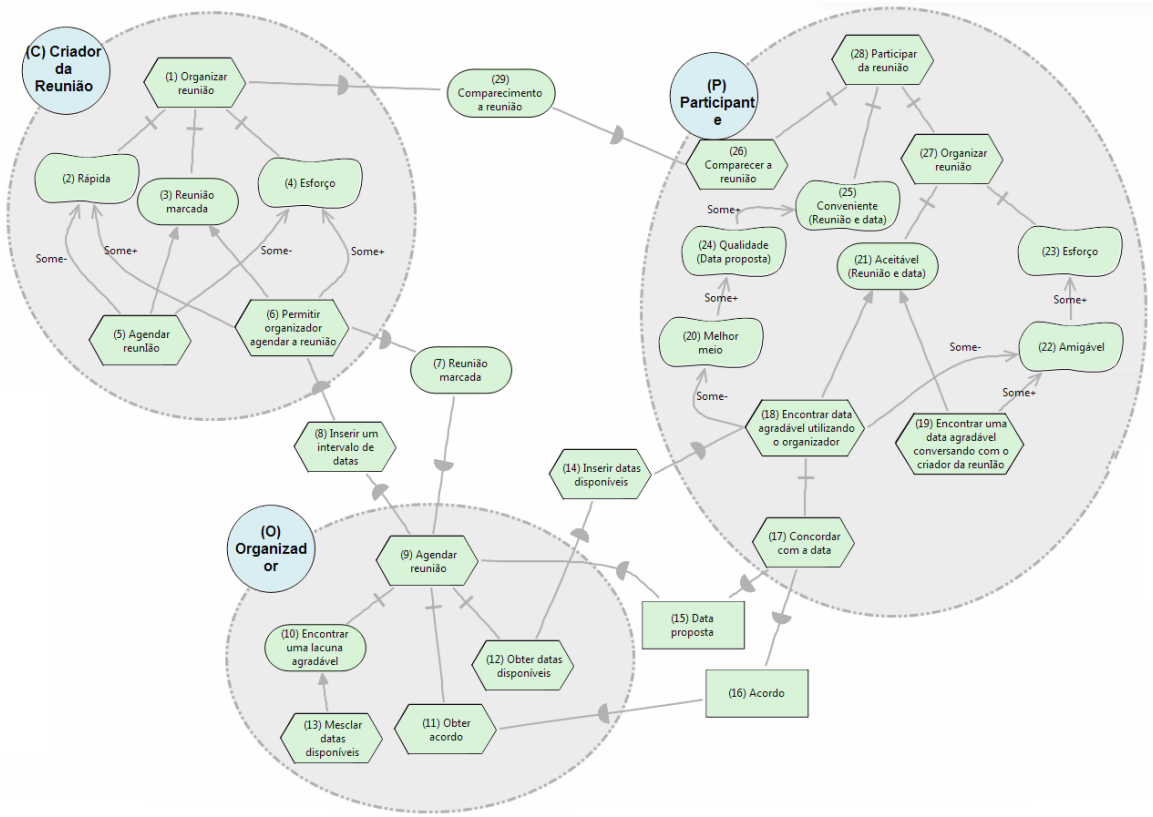


2.4 Caso de exemplo

A Figura 4 apresenta um modelo i* baseado em [16]. Trata-se de um caso de organizador de reuniões, onde um responsável deverá determinar a data e o local da reunião para os demais participantes. O modelo está dividido em três atores com suas respectivas necessidades e atividades. Como forma de identificação, foram acrescentados para esse modelo, numerações para os elementos intencionais, e siglas para os atores do sistema.

- **Criador da reunião:** Deve perguntar a todos os participantes sobre as suas respectivas disponibilidades para a reunião durante um intervalo de tempo, baseado nas agendas pessoais. Esse intervalo de tempo contém um conjunto de datas excluídas de acordo com as respostas dos participantes.
- **Organizador de reunião:** Propõe uma data que não esteja no conjunto de datas excluídas e deve estar de acordo com as datas de preferência dos participantes.
- **Participante da reunião:** Devem concordar com a data de reunião uma vez que a data aceitável seja encontrada.

Figura 4: Representação do modelo i* referente ao exemplo do organizador de reuniões.

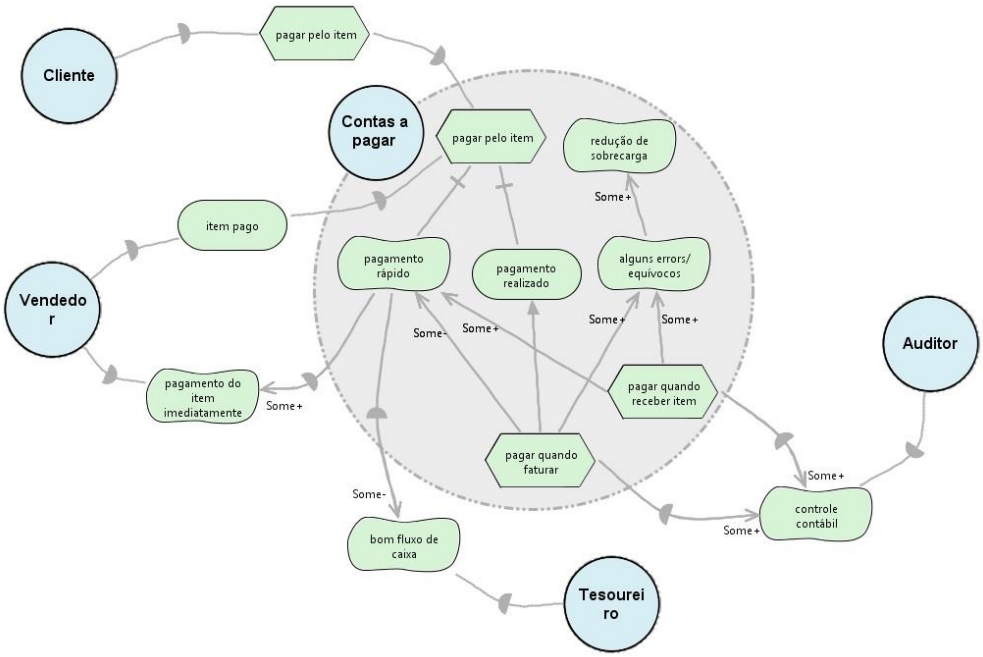


2.5 Análise Cross-impact

A análise *cross-impact* constitui na compreensão do valor semântico das ligações de dependência do modelo i*. Os modelos SR facilitam na identificação da análise *cross-impact* através do uso de ligações *means-ends* para *softgoals* [16]. Inicialmente, foi utilizado apenas para compreender as metas afetadas a partir de um meio.

A Figura 5 apresenta um exemplo da análise *cross-impact*, através de um sistema de contas a pagar. O ator cliente depende do sistema para "pagar pelo item". O ator fornecedor depende da ação do cliente junto ao sistema, para alcançar o objetivo "item pago". Além disso, o fornecedor também depende do sistema para que o pagamento seja realizado imediatamente. O ator tesoureiro depende do sistema para obter um bom fluxo de caixa. Por fim, o ator auditor depende do sistema para o controle contábil. Para esse caso, podemos ver que as ligações do *softgoals* são afetadas pelas ações do ator "Contas a pagar". As *tasks* "pagar quando receber item" e "pagar quando faturar", deverão afetar a *softgoal* "controle contábil" através da ligação de dependência, que pode ser representada semanticamente como uma ligação *contribution*.

Figura 5: Abordagem *cross-impact*.

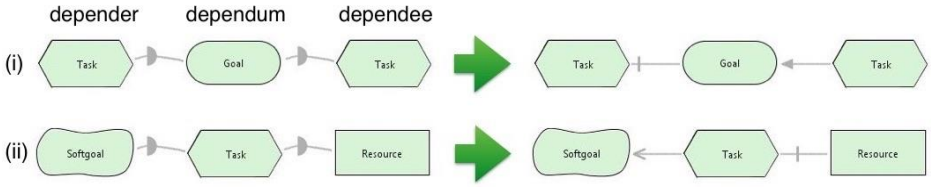


A análise *cross-impact* deve ser aplicada para cada ligação entre os elementos e atores, e é parte crucial para as transformações propostas neste trabalho.

De acordo com a Figura 5, é possível garantir uma relação de *cross-impact* com os elementos e ligações de dependências. Assim, realizamos uma análise das ligações para os demais elementos intencionais, como *goal*; *softgoal*; *resource*; e *task*.

Para cada ligação dos elementos do i* foi realizado uma análise semântica das ligações de acordo com as regras do modelo SR que estão baseadas na análise *cross-impact*. A Figura 6 apresenta apenas duas transformações baseadas na análise semântica. A análise possui um total de 64 combinações de ligação, considerando 4 elementos possíveis para o *depender*, 4 elementos para o *dependum* e os 4 elementos possíveis para o *dependee*.

Figura 6: Aplicação das regras de transformação dos conflitos semânticos



No primeiro caso (elemento i da Figura 6), temos os elementos intencionais relacionados com a ligação do modelo SD, onde a *task* representa o elemento interno do *depender*, o *goal* representa o elemento *dependum*, e por fim, a outra *task* representa o elemento interno do ator *dependee*. A transformação poderá substituir as ligações de dependência do modelo SD, por ligações do modelo SR, de acordo com as regras semânticas. Assim, a ligação entre a *task* e o *goal*, sendo estes elementos do *depender* e *dependum* (nessa ordem), poderão ser substituidos por uma ligação *task-decomposition*, enquanto a ligação entre a *goal* e *task*, como *dependum* e *dependee*, será uma ligação *means-ends*.

Para o segundo caso (elemento ii da Figura 6), o *softgoal* deverá representar o elemento interno do *depender*, a *task* representa o *dependum*, e por fim, o *resource* representa o outro elemento interno do *dependee*. A análise da Figura 6 é sempre realizada da esquerda para a direita. A ligação entre o *softgoal* e a *task*, poderá ser substituida por uma ligação *contribution*, enquanto a ligação entre a *task* e o *resource*, será substituida por uma ligação *task-decomposition*.

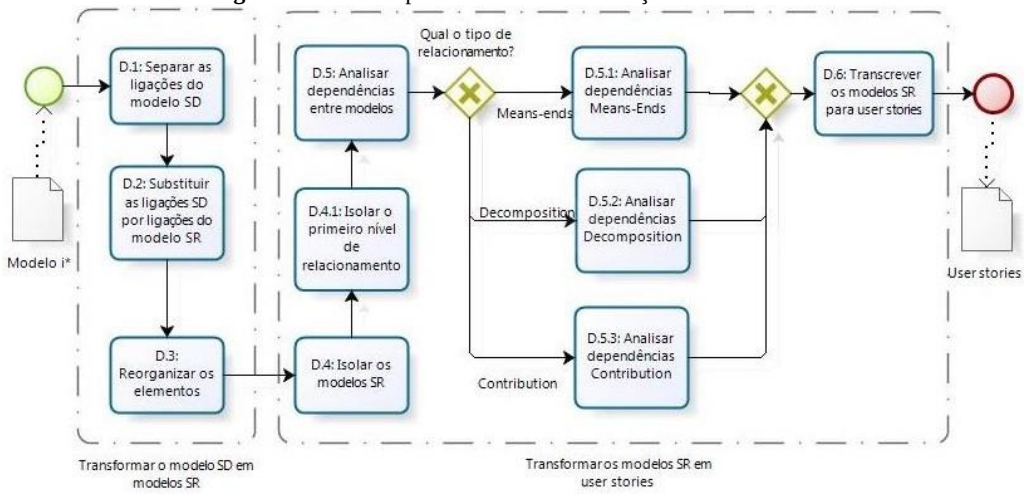
3 Proposta das diretrizes

Um conjunto de diretrizes são utilizadas para transformar o modelo i* em *user stories*. Assim, é possível estabelecer uma documentação para os métodos ágeis, baseada nas razões, intenções e requisitos não funcionais.

As diretrizes são divididas em duas etapas que foram propostas neste trabalho. A primeira etapa do processo baseia-se na transformação dos modelos SD em SR, para gerar um conjunto de modelos temporários, que deverão servir como entrada para a fase posterior. A segunda etapa utiliza o resultado da primeira etapa para transformar os modelos gerados em *user stories*.

A Figura 7 apresenta uma visão completa do fluxo do processo através da notação de modelagem de processos de negócio (BPMN). O processo apresenta um conjunto de diretrizes separadas em macro atividades: transformar modelos SD em modelos SR; e transformar modelos SR em *user stories*. Cada macro atividade é representada na Figura 7 como um agrupamento (região pontilhada) das atividades.

Figura 7: Fluxo do processo utilizando notação BPMN



As seguintes diretrizes serão apresentadas para a primeira fase do processo "transformar modelos SD em modelos SR":

- **D.1:** Separar as ligações do modelo SD em *requirements path*
- **D.2:** Substituir as ligações SD por ligações SR
- **D.3:** Identificar o início de cada *requirement path*

A segunda fase "transformar modelos SR em *user stories* "deverá realizar as transformações de acordo com as seguintes diretrizes:

- **D.4:** Isolar os modelos SR, de cada *requirement path*
  - **D.4.1:** Cada modelo deve ser isolado considerando apenas o primeiro nível de relacionamento
- **D.5:** Analisar as dependências entre os modelos
  - **D.5.1:** Analisar dependências *Means-Ends*
  - **D.5.2:** Analisar dependências *Decomposition*
  - **D.5.3:** Analisar dependências *Contribution*
- **D.6:** Transcrever e transformar os modelos em *user stories*

3.1 D.1 - Separar as ligações do modelo SD em *requirements path*

Primeiramente, é preciso separar os relacionamentos em partes para que possam ser caracterizadas como uma especificação de requisitos. Nesse caso, cada dependência e seus respectivos relacionamentos internos devem ser analisadas para cada um dos atores. Essa estrutura é denominada *requirement path*.

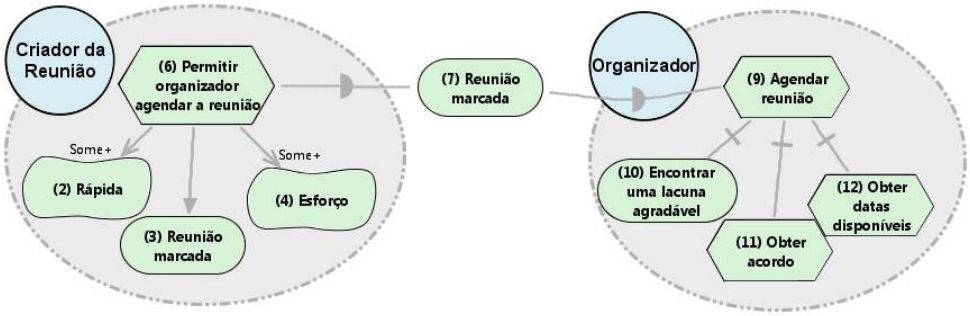
O requirement path trata-se de uma representação de um requisito considerando um fluxo baseado em um *dependum* específico, sendo este o principal elemento para identificar essas representações. Assim, um *requirement path* é composto de um *dependum* e os elementos internos associados aos atores *depender* e *dependee*, incluindo relacionamentos internos como o *means-ends*, *contribution* e *decomposition*.

Devemos separar o modelo i* em grupos de acordo com os *dependums* do modelo. Assim, os elementos separados serão agrupados em *requirement paths*. Considerando

o exemplo da Figura 4, os seguintes elementos estarão elegíveis para este agrupamento: 7 (reunião marcada); 8 (inserir um intervalo de datas); 14 (inserir datas disponíveis); 15 (data proposta); 16 (acordo); e 29 (comparecimento a reunião).

A Figura 8 apresenta a extração de um *requirement path* do modelo i* referente ao caso de exemplo da Figura 4. Assim, selecionamos apenas o grupo relacionado ao *dependum* 7 (reunião marcada) que será transformado em *user stories* posteriormente.

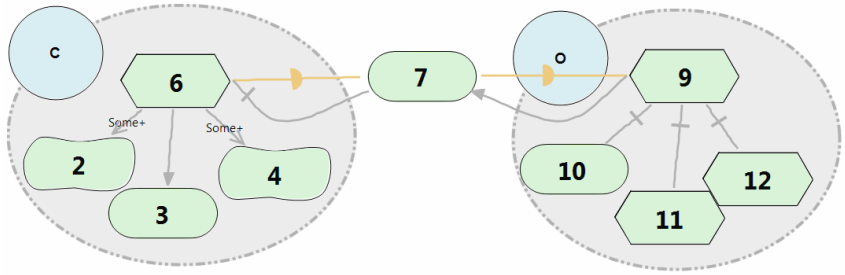
Figura 8: *Requirement path* referente ao *dependum* 7 (Reunião marcada)



3.2 D.2 - Substituir as ligações SD por ligações SR

Após separar o elemento em um *requirement path*, é preciso substituir as ligações dos relacionamentos de dependência (do modelo SD) pelos conectores do modelo SR, através do conceito apresentado pela análise *cross-impact*. As ligações do modelo SD devem ser substituídas pelas ligações do modelo SR, para facilitar a compreensão dos requisitos gerados pelo i*. A Figura 9 apresenta o resultado dessas alterações para facilitar a compreensão e a construção das *user stories*.

Figura 9: Transformação do modelo i*, após a aplicação das diretrizes D.1 e D.2



3.3 D.3 - Identificar o início de cada *requirement path*

Por último, os modelos são reorganizados verticalmente para melhorar na compreensão da abordagem e em seguida identificar o início de cada *requirement path*. Para o caso da Figura 9, é possível identificar onde um *requirement path* inicia e como ele deverá afetar os demais elementos. Assim, o caso deve iniciar a partir da ação de número 9 (Agendar reunião) pelo ator "O" (Organizador da reunião).

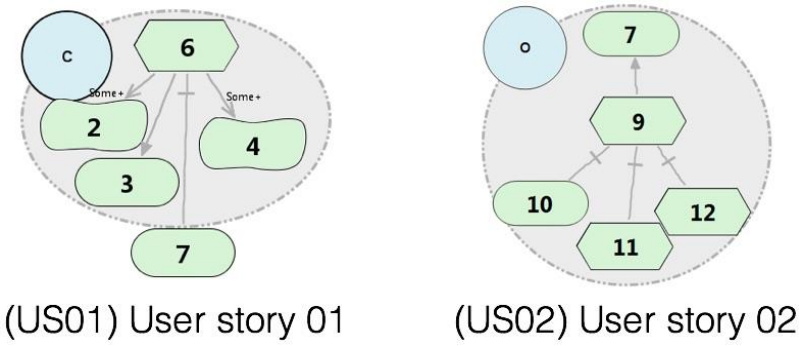
3.4 D.4 - Isolar os modelos SR, de cada *requirement path*

Com essa estrutura aplicada pelas diretrizes anteriormente citadas, é possível realizar a segunda parte da transformação. Para isso, é preciso isolar os modelos SRs e seus respectivos relacionamentos. Nesse caso, cada *requirement path* deverá ser dividido em pequenas partes. Cada parte extraída deverá gerar uma *user story*. Os demais elementos serão caracterizados como uma nova *user story* e uma relação de dependência com a *user story* existente.

A diretriz D.4.1 deve considerar apenas no primeiro nível de relacionamento. Assim, deverão apenas fazer parte do relacionamento de um nível: *means-end*, apenas quando se tratar do elemento "*means*"; *decomposition*, apenas quando o elemento analisado estiver sendo decomposto; *contribution*, apenas quando o elemento analisado for o agente contribuinte. A diretriz D.4.1 procurar limitar-se apenas ao primeiro nível para que possa ser gerado modelos mais simples possíveis. Isso deve resultar em user stories simples, com o foco apenas em uma ação, objetivo, recurso ou requisitos não funcionais. Isso torna o modelo i* legível e compreensível para a proposta.

A Figura 10 apresenta a transformação do *requirement path* após isolar os elementos em até um nível de relacionamento. A partir de então, teremos duas representações do modelo SR, do *framework* i*, que deverão transformar-se em *user stories*. O elemento 7 (reunião marcada) do modelo i* US01, não está contido no ator "C" (criador da reunião) pois o mesmo pertence ao ator "O" (organizador da reunião). Essa representação apenas demonstra que há uma dependência entre as duas *user stories*.

Figura 10: Transformação do modelo i*, após a aplicação das diretrizes D.4 e D.4.1



3.5 D.5 - Analisar as dependências entre os modelos

Analisar as dependências entre os modelos de acordo com os elementos que estão sendo utilizados, ou seja, se o elemento analisado estiver vinculado a um outro modelo SR isolado, então teremos uma dependência.

De acordo com a Figura 10, temos que o modelo i* US01 depende que o ator "O" (organizador da reunião) do modelo US02, alcance o objetivo 7 (reunião marcada), para que possa então, realizar a tarefa 6 (permitir organizador agendar reunião).

3.6 D.6 - Transcrever e transformar os modelos em *user stories*

Por fim, devemos transcrever os modelos i* em *user stories*, que deverão ser utilizadas no desenvolvimento de sistemas. Além disso, esses artefatos estarão transcritos em uma estrutura denominada story cards.

3.7 Construção de uma estrutura de Story Card

As *story cards* contém uma pequena descrição de uma funcionalidade de valor para um *stakeholder* [4]. Além disso, as *story cards* podem conter testes de aceitação, entre outras informações relevantes para a equipe de desenvolvimento. Para este trabalho, optamos pela estrutura tradicional, adicionando dois aspectos: as dependências entre as *user stories*; e a relação com os requisitos não funcionais. A estrutura de uma *story card* baseia-se nos conceitos de [4] e [7], e possuem as seguintes informações:

- **ID:** Identificador da *user story*, geralmente caracterizado por uma numeração ou código (ex.: US01.1)
- **Descrição:** Apresenta a descrição da *user story*, no formato padrão (ex.: "como <papel> eu quero <ação> para <meta>").
- **Critérios de Aceitação:** Realiza a validação da *user story*. Se todos os testes de aceitação estiverem de acordo com o que foi solicitado, então a *user story* também estará de acordo (Campo opcional)
- **Dependências:** Apresenta dependências entre as *user stories* (Campo opcional)
- **Requisitos não funcionais:** Apresenta informações sobre as metas-flexíveis (*softgoals*) ou requisitos não funcionais (Campo opcional)

Os quadros Quadro 1 e Quadro 2, apresentam as transformações dos modelos US01 e US02 respectivamente.

Quadro 1: Modelo US01 transcrito em forma de *user story*

ID: US01
Descrição: Como um "criador da reunião", eu quero "permitir organizador agendar a reunião", para obter uma "reunião marcada"
Critérios de Aceitação: • precisa ter uma "reunião marcada"
Dependências: Depende de US02, para alcançar o objetivo "precisa ter uma "reunião marcada"
Requisitos não funcionais: • Contribui com em ser "rápida" • Contribui com o "esforço"

Quadro 2: Modelo US02 transcrito em forma de *user story*

ID: US02
Descrição: Como um "organizador", eu quero "agendar reunião", para obter uma "reunião marcada".
Critérios de Aceitação: • Alcançar o objetivo "Encontrar uma lacuna agradável" • Realizar a atividade "obter acordo"

- Realizar a atividade “obter datas disponíveis”

4 Discussão

A partir do resultado deste trabalho, é possível compreender que conjunto das diretrizes de transformação do modelo i* para *user stories* é uma abordagem complementar e sem obrigatoriedade com a metodologia ágil. Assim, a abordagem auxilia na documentação ágil para mitigar os desafios encontrados, como o impacto e as constantes mudanças, além da construção de requisitos desnecessários.

Um projeto que utiliza Scrum, por exemplo, pode ser dividido em fases para que inicialmente seja elicitado requisitos a partir do modelo i*, e em seguida, gerar *user stories*. A abordagem apenas contribui e auxilia na construção de *user stories* com uma visão de alto nível, baseada em uma modelagem focada nos objetivos organizacionais, razões, intenções e requisitos não funcionais.

As limitações do modelo estão focadas nos elementos que não conseguem ser mapeado pelos *requirement paths*. Nesse caso, esses elementos não poderão estar presentes nas *user stories*, pois não caracterizam dependências e estão incorporados dentro das limitações dos atores, ou seja, são ações internas, e precisam ser estudadas com cautela pelos analistas de negócios, sobre a sua necessidade.

5 Trabalhos Relacionados

Nesta seção, apresentamos os trabalhos relacionados com a proposta. Esses trabalhos foram utilizados como meio para compreender a problemática e aplicar as regras de transformação do modelo i* em *user stories*.

[6] utilizou um conjunto de heurísticas para enriquecer as *user stories* transformando as mesmas em modelos i*. Os modelos i* contribuem como uma forma de documentação dos requisitos no ambiente ágil que podem ser visualizados de forma abrangente com seus relacionamentos de acordo com o ambiente de negócios. A principal preocupação está relacionada com a falta de documentação no ambiente de desenvolvimento ágil, pois isso é um dos principais desafios da metodologia.

[17] utilizou um conjunto de heurísticas para transformar modelos i* em *user stories*, incluindo informações que antes não eram capturadas pelos métodos ágeis, como os requisitos não funcionais, as razões, motivações e intenções dos papéis (ou atores) com suas respectivas metas e ações. A limitação do trabalho apresentou ausência dos casos de exemplo, e utilizou apenas um estudo de caso em ambiente controlado. Isso resultou em uma abordagem limitada, para a evolução da técnica proposta.

[18] propõe uma nova linguagem de modelagem, denominada Scrum i*, para projetos que utilizam métodos ágeis, em especial o Scrum. Esta nova linguagem visa simplificar a técnica i*, para ser integrada ao Scrum. Além disso, procura avaliar a viabilidade, benefícios e desafios para adoção da nova linguagem de modelagem em um ambiente industrial. Mesmo tratando-se da transformação do modelo i*, obtemos

apenas uma abordagem que foca na metodologia de um projeto, enquanto que a abordagem deste trabalho está focada na documentação e nos artefatos gerados.

6 Considerações Finais e Trabalhos Futuros

A documentação ágil ainda não definiu qual a melhor forma de documentar e compartilhar informações de forma eficiente. Em muitos casos, a falta de uma documentação necessária, que possua razões, intenções e requisitos não funcionais para o sistema proposto pode ocasionar em uma documentação incompleta, além de construir requisitos desnecessários e aumentar o impacto das mudanças desses requisitos.

A metodologia ágil possui alguns desafios na fase de documentação, o que pode causar problemas na comunicação e na qualidade do produto [6], assim como o aumento do escopo, do custo e do tempo. Outro desafio está relacionado com a eficácia da comunicação entre as partes interessadas, que depende de diversos fatores como a disponibilidade do cliente, o consenso entre os grupos de clientes e a confiança entre o cliente e os desenvolvedores, principalmente durante a fase inicial de um projeto [3]. Dessa forma, uma abordagem baseada em *design rationale*, pode fornecer requisitos mais compreensíveis para o desenvolvimento de software, através de uma modelagem que possa fornecer uma visão geral do sistema a ser desenvolvido.

Este trabalho apresenta uma proposta de transformação do modelo i* em *user stories* para produzir uma documentação baseada nas razões, intenções e requisitos não funcionais. A partir da identificação do problema, foi possível constatar a necessidade da aplicação das diretrizes de transformação da proposta. Assim, trata-se de uma proposta complementar para a Engenharia de Requisitos, integrando o conceito de *design rationale*, representado pelo framework i*, com a documentação ágil.

Os modelos GORE podem ser facilmente relacionados com projetos ágeis, sendo incluídos como uma abordagem complementar, para evitar constantes mudanças nos requisitos ou o desenvolvimento de funcionalidades desnecessárias para o projeto. Além disso, a modelagem GORE pode ser utilizada como uma documentação prévia dos projetos ágeis. Isso implica que os métodos ágeis podem estabelecer um padrão de documentação que contribua na compreensão dos requisitos do sistema. Essa documentação prévia é denominada *design rationale*.

Como trabalhos futuros, podemos destacar as seguintes atividades: (i) Construção de uma solução automatizada, através de uma linguagem específica de domínio (DSL, ou *Domain Specific Language*) para realizar uma transformação de acordo com os modelos i*. Essa solução poderá contribuir com os analistas de negócios e os *stakeholders* na construção de um produto com um grau de certeza das funcionalidades produzidas; (ii) Aplicar a proposta em casos reais, utilizando times de desenvolvimento de software especializados em métodos ágeis. Para isso, um conjunto de pesquisas deverão ser realizadas através de questionários e estudos de campo, utilizando a solução automatizada; (iii) A proposta de transformação ainda possui uma visão limitada diante das possíveis variações da especificação dos requisitos não funcionais. Assim, um estudo aprofundado da semântica e sintaxe do modelo i*, focado

nos requisitos não funcionais deverá ser realizado para contribuir com as diretrizes de transformação do modelo i* para *user stories*.

References

1. Agile Manifesto (2001), http://www.manifestoagil.com.br/, [Online; accessado em 01/09/2015]
2. Alencar, F., Marin, B., Giachetti, G., Pastor, O., Castro, J., Pimentel, J.: From i* requirements models to conceptual models of a model driven development process. Lecture Notes in Business Information Processing pp. 99–114 (2009)
3. Cao, L., Ramesh, B.: Agile requirements engineering practices: An empirical study. Software, IEEE 25(1), 60–67 (2008)
4. Cohn, M.: User Stories Applied: For Agile Software Development. Addison-Wesley Professional (2004)
5. Hoda, R., Noble, J., Marshall, S.: How much is just enough?: Some documentation patterns on agile projects. In: Proceedings of the 15th European Conference on Pattern Languages of Programs. pp. 13:1–13:13. EuroPLoP '10 (2010)
6. Jaqueira, A., Lucena, M., Alencar, F.M.R., Castro, J., Aranha, E.: Using i * models to enrich user stories. In: iStar'13. pp. 55–60 (2013a)
7. Jeffries, R.: Essential xp: Card, conversation, confirmation. http://goo.gl/2e8VXG (2001), [Online; accessado em 01/09/2015]
8. López, C., Cysneiros, L.M., Astudillo, H.: Ndr ontology: Sharing and reusing nfr and design rationale knowledge. In: Managing Requirements Knowledge, 2008. MARK '08. First International Workshop on. pp. 1–10 (2008)
9. Prause, C.R., Durdik, Z.: Architectural design and documentation: Waste in agile development? In: Software and System Process (ICSSP), 2012 International Conference. (2012)
10. Read, A., Briggs, R.O.: The many lives of an agile story: Design processes, design products, and understandings in a large-scale agile development project. In: System Science (HICSS), 2012 45th Hawaii International Conference on. pp. 5319–5328 (2012)
11. Santander, V., Castro, J.: Deriving use cases from organizational modeling. In: Requirements Engineering, 2002. Proceedings. IEEE International Conference on. pp. 32–39 (2002)
12. Santander, V.F.A., Castro, J.: Developing use cases from organizational modeling. In: WER. pp. 246–265 (2001)
13 Sauer, T.: Using design rationales for agile documentation. In: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on. pp. 326–331 (2003)
14. Sen, A., Hemachandran, K.: Elicitation of goals in requirements engineering using agile methods. In: Computer Software and Applications Conference Workshops (2010)
15. Stettina, C.J., Heijstek, W.: Necessary and neglected?: An empirical study of internal documentation in agile software development teams. SIGDOC '11. pp. 159–166. (2011)
16. Yu, E.S.K.: Modelling Strategic Relationships for Process Reengineering. Ph.D. thesis, University of Toronto (1995)
17. NETO, T. C. de L. Enriquecendo as Histórias de Usuários com a Abordagem i*. 2014
18. SCHEIDEGGER, M. E. dos Santos de. Integrando Scrum e a Modelagem de Re- quisitos Orientada a Objetivos por meio do SCRUM i*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, Recife, PE, 2012.