5
Trabalhos Relacionados

Este capítulo apresenta uma análise de algumas propostas sobre a modelagem intencional e a transparência de software base de nosso trabalho.

5.1.Modelagem Intencional Casos Práticos

Dentro de nosso trabalho temos como proposta a criação de um protótipo baseado nos modelos intencionais, portanto tivemos a necessidade de pesquisar trabalhos que utilizaram este tipo de modelagem e os mecanismos que eles utilizaram para passar dos modelos até a implementação. No Capítulo 2 apresentamos as diferentes abordagens de modelagens usadas em jogos educacionais para ensino na engenharia de software, identificamos que alguns casos não foi usado nenhum método ou não possuíam suficiente informação relacionada com esta parte do processo. Nosso foco nesta sessão é explorar a metodologia intencional e identificar aspectos uteis das experiências aqui relatadas para o nosso trabalho no SimulES.

5.1.1.Estendendo Tropos para uma Implementação em Prolog: um Estudo de Caso Usando o Problema do Agente Coletor de Alimentos (FCAP) [49]

A metodologia Tropos é centrada nos requisitos e no *Framework i** que permite reduzir incompatibilidade entre o sistema e seu ambiente. Esta possui ator, agente, posição, papel e as dependências entre eles; permitindo dependências por metas, tarefas e recursos.

Estes conceitos não somente são usados na produção de requisitos, mas também são usados nas fases de arquitetura e desenho detalhado.

A abordagem do artigo é orientada ao Tropos e para a implementação utiliza-se Prolog.

As fases do método Tropos são descritos de forma geral:

(i) **Requisitos iniciais:** cujo objetivo é o entendimento do problema a partir da análise da organização. Nesta fase se faz o modelo de dependência estratégica (SD) para descrever os relacionamentos entre atores e o modelo de raciocínio estratégico (SR) para identificar as estratégias internas de cada ator. Para o caso FCAP (pelas siglas em inglês de *Food Collecting Agent Problem*) esta fase é omitida, pois não se tem um contexto social.

(ii) **Requisitos finais:** onde é incluído explicitamente o sistema com o seu ambiente operacional, a partir dos requisitos funcionais (metas) relevantes e requisitos de qualidade (metas flexíveis), nesta fase também se identificaram os agentes do *grupo coletor* e *provedor de alimentos*, suas dependências são restrições de comportamento como se mostra na seguinte Figura.

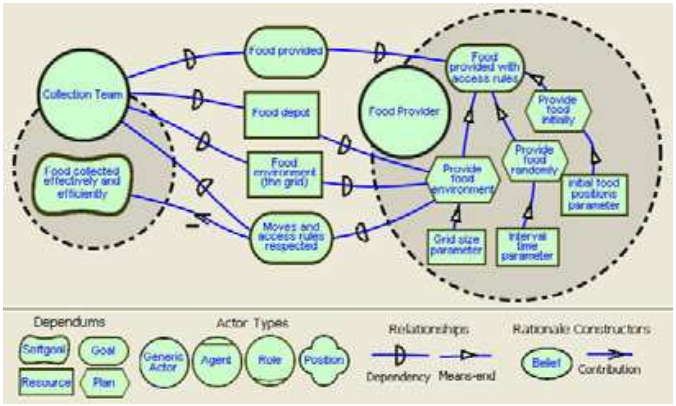

Figura 34 – Identificação de atores genéricos para FCAP [49].

(iii) **Arquitetura:** o objetivo desta fase é a definição global da arquitetura em termos de subsistemas e suas dependências, modelo da arquitetura do sistema e modelagem em partes pequenas para que sejam facilmente gerenciáveis, desta forma se descreve como os agentes funcionam através de seus papéis. Nesta fase para FCAP identificaram-se novos agentes, suas principais capacidades e cada um dos papéis para os agentes. Também foi definida a posição *teamMember* para representar todos os membros da equipe.

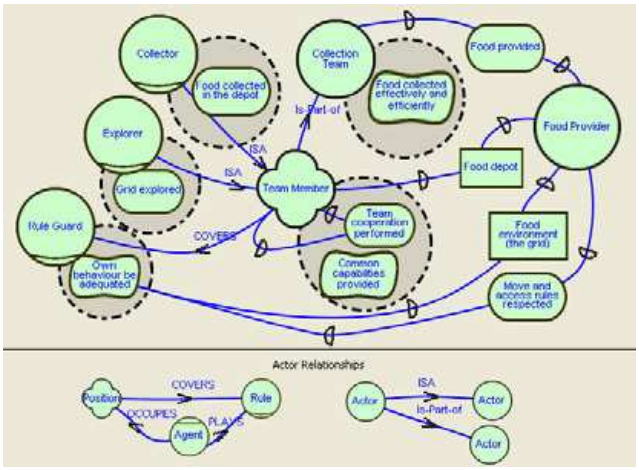

Figura 35 – Descrevendo papéis de agentes em FCAP [49].

(iv) **Desenho:** é refinado cada componente arquitetural através de comunicação entre agentes, mecanismo de transporte de mensagens, ontologias e protocolos. Produz-se o diagrama de classes de agentes, diagrama de interações, diagrama de capacidades e diagrama de planos.

Para FCAP cada agente foi decomposto e foram especificadas suas capacidades como também suas crenças, foram também atualizadas as dependências para FCAP (omitiu-se as contribuições de metas flexíveis entre os atores para dar simplicidade a o diagrama).

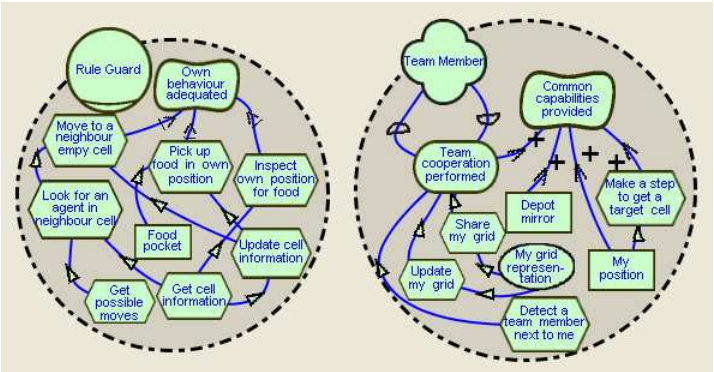

Figura 36 – Decomposição de cada agente FCAP [49].

(v) **Implementação:** Tropos propõe um diagrama de atividades, mas para FCAP, foram usadas seqüências de cenários.

Em Prolog foram implementadas as seqüências acima citadas, pois provém metas lógicas e raízes de planos, também se propuseram quatro atributos de implementação chamados *begin, at end, at call and always.*

O mapeamento do desenho até a implementação foi feito através da instanciação dos predicados (agentes, papéis, posições), ou seja, os relacionamentos definidos em Tropos.

Este método mostra explicitamente o levantamento de requisitos feito através do *framework i**.

A solução proposta utiliza uma técnica de levantamento de requisitos através de cenários, sendo esta uma boa prática que permite justificar os elementos que estariam presentes tanto na modelagem como na implementação. Alem disso, este trabalho mostra a instanciação dos elementos modelados até a implementação.

5.1.2.“Tropos: uma Metodologia de Desenvolvimento de Software Orientada a Agentes” [50]

Baseado em Tropos foi desenvolvida uma aplicação nomeada *eCulture* para o governo de Trentino (Provincia Autónoma de Trento, o PAT) na qual um agente Web fornece informações e serviços culturais da PAT para cidadãos e turistas que procuram coisas para fazer, e para os estudantes que procuram material relevante relacionado a estudos. As informações fornecidas são obtidas de museus, exposições, organizações culturais entre outros.

(i) **Requisitos iniciais:** neste ponto se identificou e analisou as partes interessadas (interessados) e suas intenções. Os interessados foram modelados como atores sociais. As dependências entre eles foram modelados visando que: objetivos sejam alcançados, tarefas sejam realizadas e recursos utilizados. As intenções modeladas nesta etapa através de uma análise orientado a metas serão decompostas ou refinadas em fases posteriores.

Como resultado desta atividade na Figura 37 é apresentado o diagrama de atores do contexto de *eCulture.* O ator cidadão é associado a uma meta concreta "obter informações culturais" e turistas tem associada a meta flexível "desfrutar da visita". No mesmo sentido, PAT quer aumentar o uso da internet enquanto o museu pretende oferecer serviços culturais. Finalmente o diagrama ilustra a dependência da meta flexível onde o cidadão depende de PAT para que *impostos sejam bem gastos.*

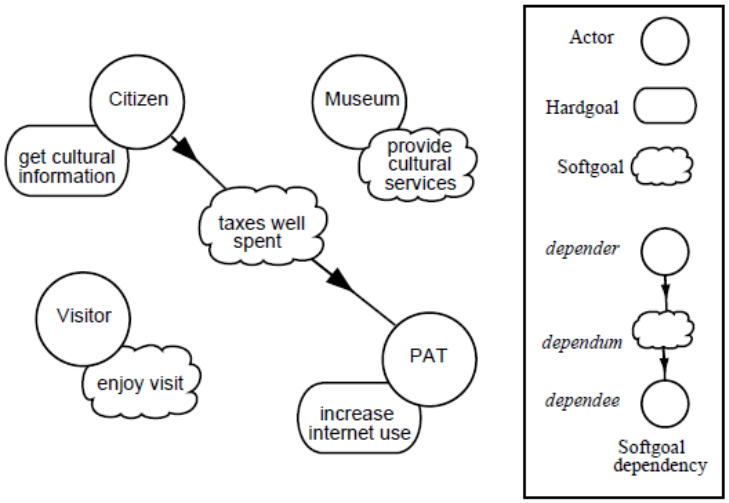

Figura 37 – Diagrama de atores, modelando os *stakeholders* do projeto *eCulture* *[50].*

Já na Figura 38 os autores descrevem que a meta "obter informação cultural" do ator Cidadão é descomposta em "visitar instituições culturais" e "visitar sitios Web culturais". Estas duas sub-metas podem ser vistas como maneiras alternativas do cumprimento da meta principal.

A decomposição de uma meta pode ser feita através do relacionamento *meios-fim* e sua análise tem como objetivo identificar as tarefas, recursos e metas flexíveis que fornecem meios para atingir a meta. Neste trabalho, a tarefa "visitar o sistema *eCulture*" é um meio para cumprir a meta "visitar sítios Web culturais" esta tarefa é decomposta em dois sub-tarefas nomeadas "usar sistema *eCulture*" e "acesso à internet" os quais são as razões para o conjunto de dependências entre cidadão e PAT: "Sistema *eCulture* esteja disponível", "infra-estrutura da internet

esteja disponível" e "Sistema *eCulture* esteja usável". A análise do visitante pode ser mais simples "o plano para visitar" pode dar uma contribuição positiva para a meta flexível "desfrutar a visita" e para isso o visitante precisa que o "Sistema eCulture esteja disponível".

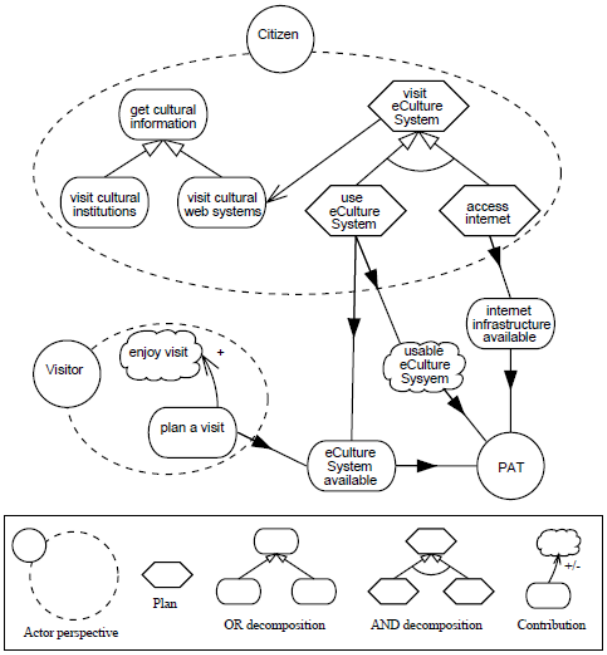

Figura 38 – Diagrama de metas para Cidadão e Visitante. Observe se a meta e plano de decomposição, a análise *meios-fim* e a contribuição positiva à meta flexível [50].

(ii) **Requisitos finais:** incide sobre o que *o sistema a ser* ou *system-to-be*, ou seja, o sistema *eCulture* no seu ambiente operacional, junto com suas funções e qualidade relevante. Por tanto, *sistema a ser* é representado como um ator que tem dependências com os atores da organização, estas dependências são as que definirão os requisitos funcionais e não funcionais do sistema.

O diagrama de atores na Figura 39 representa o sistema *eCultura* e apresenta um conjunto de metas concretas e metas flexíveis que PAT (Provincia Autónoma de Trento) delega para ele**.**

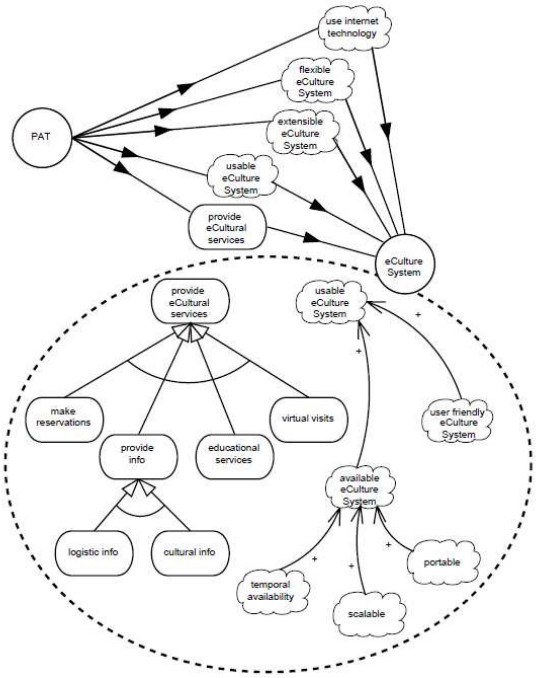

Figura 39 – Segmento do diagrama de atores incluindo PAT e o sistema *eCulture* e o diagrama de metas do sistema *eCulture* [50].

(iii) **Arquitetura:** nesta fase é definida a arquitetura global do sistema em termos de subsistemas (atores) interligados através dos dados e os fluxos de controle (dependências).

Esta fase é decomposta em três etapas. Na primeira etapa foram identificados novos atores após uma análise das metas do sistema, da escolha da arquitetura, padrões e estilo, e da inclusão de atores que contribuem positivamente para a realização dos requisitos funcionais e não funcionais. Após finalizada a primeira etapa o diagrama de atores é estendido, onde, o diagrama de atores produzido é analisado em detalhe para identificar as capacidades necessitadas pelos atores para exercer suas metas e tarefas. O último passo consiste em definir um conjunto de *tipos de agentes* e atribuir a cada um eles um ou mais recursos diferentes.

(iv) **Desenho:** nesta fase se lida com a especificação a nível micro dos agentes. Ou seja, metas, crenças e capacidades dos agentes assim como também a comunicação entre eles são especificadas. Além disso, segundo descrevem os autores em [50] esta etapa está

relacionada com as escolhas da implementação. Eles representaram as capacidades e planos dos agentes usando diagramas de atividades UML e diagramas AUML [69] para especificar os protocolos dos agentes.

(v) **Implementação:** o ambiente de desenvolvimento orientado a agente usado foi JACK e está integrado à linguagem Java. Os agentes em JACK são componentes de software autônomos com metas explicitas a serem executadas, além disso, são programados com um conjunto de tarefas (planos) a fim de tornar-los capazes de atingir as metas. Conforme os autores, as especificações de seções anteriores tem correspondência com a implementação em JACK, pois neste ambiente de programação é possível definir agentes, capacidades, crenças, eventos e tarefas (planos) segundo é apresentado na Figura 40 que ilustra um fragmento do ambiente Jack para o sistema eCulture.

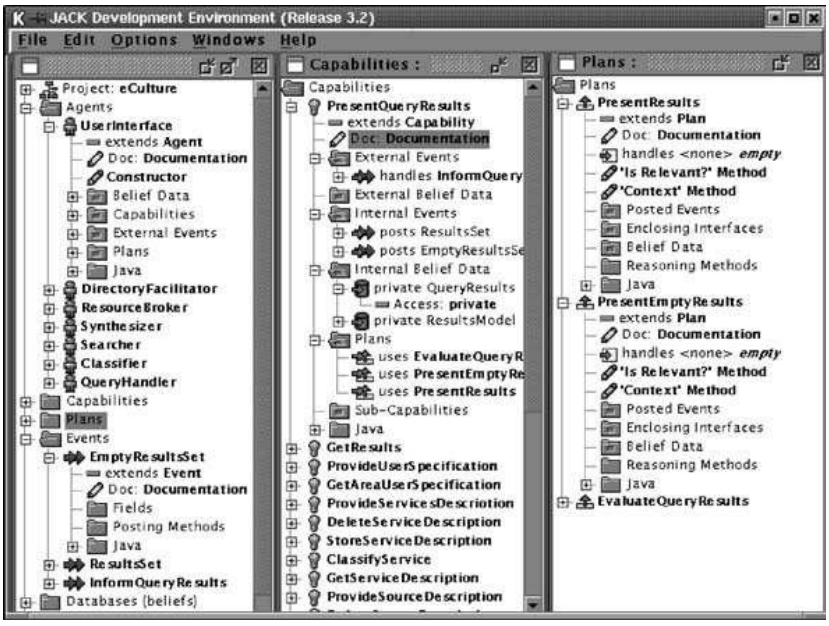

Figura 40 – Ambiente de desenvolvimento JACK para projeto *eCulture.*

5.1.3.“Uma Abordagem para Modelagem Intencional de Avaliação de Riscos de Segurança em *Web Services*” [51]

O aumento nos últimos anos do comércio eletrônico e serviços bancários eletrônicos estão levando com que os principais fornecedores de pagamentos eletrônicos fiquem preocupados com oferecer a seus clientes garantia e segurança de suas informações pessoais as quais são fornecidas pelos clientes de forma *online*. É neste contexto que os autores deste trabalho usam um exemplo de uma empresa X que tem que proteger seu negócio *on-line* de venda do produto Y.

Os conceitos associados com a modelagem de dependências de atores têm suas raízes na Engenharia de Requisitos (RE por suas siglas em inglês). Os métodos de RE podem ser usados para modelar as metas organizacionais, processos, relações e atores e executar uma avaliação de risco com boa qualidade, o qual é necessário para compreender a organização de uma forma clara. Estas são as justificativas conforme [51] para a escolha da modelagem intencional.

Conforme [51] *i** é baseado na Engenharia de Requisitos (RE) e pode ser considerada uma poderosa ferramenta que auxilia na modelagem de tarefas organizacionais, processos, atores e metas. Assim como também permite que os engenheiros de requisitos modelem, em detalhe, processos atuais para modificá-los, a fim de aperfeiçoar, melhorar e aumentar a produtividade da empresa. Todos esses benefícios podem ser obtidos muito cedo, mesmo quando o projeto ainda esteja por começar. *i** explora "porquê" os processos são realizados na forma existente. O Comportamento esperado do software e sua razão de ser também podem ser modelados usando i *. No entanto, *i** não “leva diretamente” em conta a precisão, integralidade e consistência como UML faz. Em contraste, *i** leva em consideração principalmente os interesses dos atores, suas metas, razões, tarefas e preocupações.

Este trabalho usa *i** para a modelagem dos requisitos e elementos de gestão de riscos para ajudar aos gestores de riscos a identificar, monitorar, analisar e controlar os riscos, tudo desde um ponto de vista de metas. *i** fornece uma análise qualitativa da viabilidade do projeto sob vários cenários. No contexto deste trabalho, essa análise permitiu verificar as ações necessárias para controlar os riscos. Ou seja, se as metas do projeto podem ser satisfeitas nos cenários

estudados. Para cada projeto, os requisitos podem ser modelados como metas concretas e metas flexíveis a serem alcançadas.

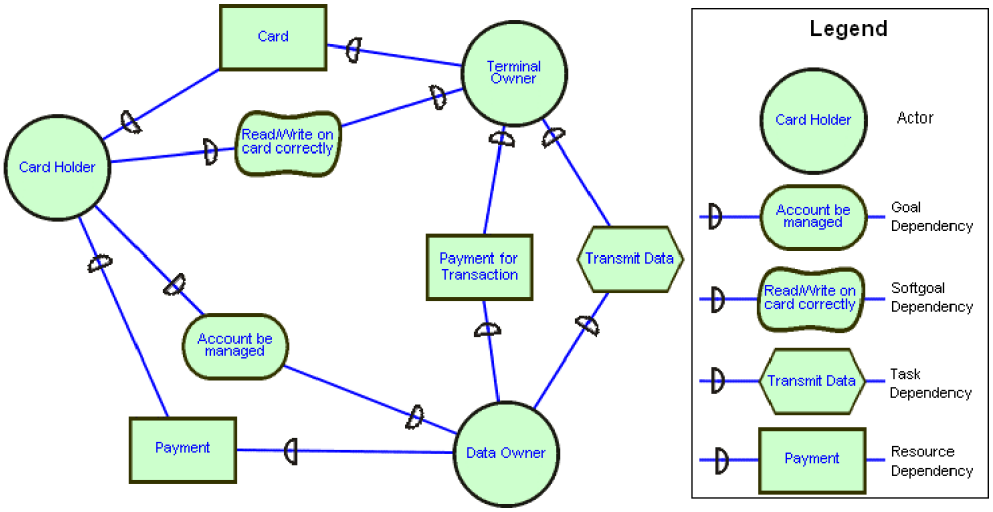

Figura 41 – Diagrama SD para o *Web Service* baseado em cartões de pagamento.

A Figura 41 representa as dependências entre os atores do sistema inteligente de cartões de pagamento que suporta os *Web Services*. A dependência por meta "Conta seja gerenciada" indica que o titular do cartão (*Cardholder*) precisa do proprietário dos dados (*Data Owner*) para gerenciar a conta. O *Proprietário dos dados,* por outro lado, espera pagamento do titular do cartão que é representada pela dependência por recurso *pagamento (*Payment*)*. Além disso, a meta flexível “Ler/Escrever no cartão corretamente” é uma dependência difícil de determinar pelo significado qualitativo de "corretamente". A dependência por tarefa "Transmissão de Dados" indica que o *proprietário dos dados* que precisa do *proprietário do terminal* (Terminal Owner) para transmitir dados; *proprietário do terminal* tem que executar a transmissão como definida pelo proprietário dos dados (Data Owner).

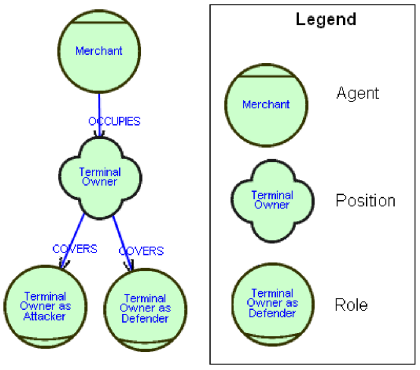

Figura 42 – Tipos de atores do *Web Service* baseado em cartões de pagamento [51].

Na Figura 42 são ilustrados os diferentes tipos de atores identificados para o sistema. A Figura 43 ilustra o diagrama de SR para papéis do sistema de pagamento. O *titular do cartão* tem uma meta interna "Comprar bens com o cartão inteligente." Se *titular do cartão* usa cartão para fazer isso, então, *titular do cartão* tem que efetuar a tarefa interna "Usar o cartão.". A meta "Comprar bens com o cartão inteligente." e a tarefa "Usar o cartão." estão ligadas com um relacionamento *meios-fim*. O *Proprietário do Terminal* tem a tarefa "Processar transação" a qual está subdividida em duas sub-tarefas "Ler / escrever no cartão" e "Ler / escrever na Base de Dados" relacionada com a tarefa "Processar transação" através do relacionamento de decomposição de tarefa. A tarefa "Ler / escrever sobre cartão" está associada com o *titular do cartão* que a sua vez está relacionada com a meta flexível externa "Ler / escrever no cartão corretamente". A tarefa externa "Ler / escrever na base de dados central" assim como também a meta flexível "Enviar dados corretamente" estão em uma dependência indo do proprietário de dados para proprietário do terminal. Contudo, uma visão mais detalhada apresenta-nos que tanto "Ler / escrever na base de dados central" e "Enviar dados corretamente" depende da tarefa interna de *proprietário de terminal* "Ler escrever sobre a base de dados". As dependências entre elementos internos e externos do diagrama SR possibilitam a realização de uma modelagem mais detalhada.

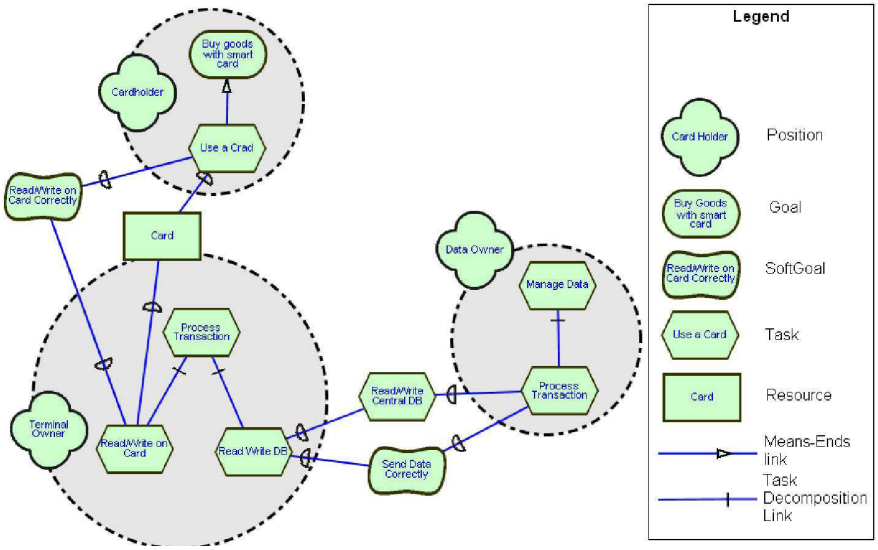

Figura 43 – Diagrama SR para o *Web Service* baseado em cartões de pagamento [51].

Na Figura 44 se representa um ataque ao sistema. Para realizar um ataque, um invasor deve explorar as vulnerabilidades; estes aproveitamentos de vulnerabilidades são representados por sub-tarefas da tarefa associada ao ataque, e que estão ligadas através da relação de decomposição de tarefas. Notamos que os elementos da tarefa correspondente à exploração de vulnerabilidades possuem uma letra "V" no lado direito do elemento, como mostrado na Figura 44.

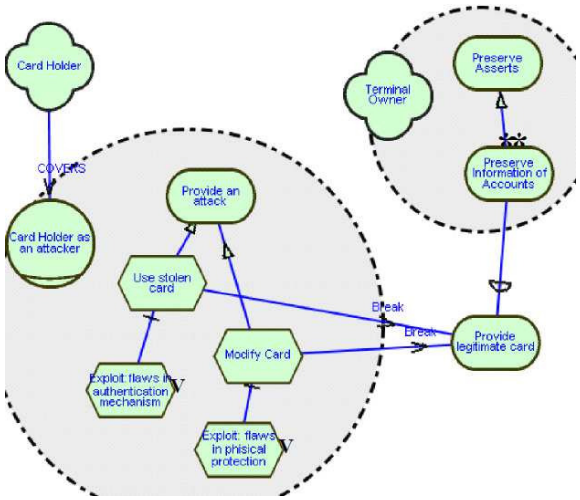

Figura 44 – Representação de ataques para o *Web Service* baseado em cartões de pagamento [51].

Analisando esta situação, se um ator depende de outro ator que é um atacante, o atacante pode valer-se de uma dependência vulnerável. Na Figura 42 as tarefas marcadas com V levam a que a dependência de *proprietário de terminal* com titular do cartão seja "quebrada" (break) e faze a vulnerabilidade existente.

5.2. Transparência de Software

Um fenômeno está ganhando momento em nossa sociedade: a **Transparência** e seu impacto direto, o direito a ser informado, está ocupando parte dos esforços da sociedade moderna para seu estudo e aplicação. Este fato reflete um requisito geral para sociedades democráticas, cada vez mais demandantes na solicitude de seus direitos.

E por que de sua importância? Destacados estudiosos do tema [9], [55], [56], [57], [58] concordam que a tendência a nível mundial fará com que a transparência seja destaque nos âmbitos sociais, governamentais e públicos, portanto, sendo o software um elemento que permeia vários destes aspectos os engenheiros de software terão que estar preparados para esta demanda. Segundo [8] se vislumbra que engenheiros terão que possuir métodos, técnicas e ferramentas para ajudar a fazer software transparente.

Focando o nosso trabalho neste contexto e ententendo que o aumento da demanda por transparência é um fato, pretendemos fazer deste trabalho com o SimulES-W um caso de estudo de como aplicar os conceitos de transparência.

O mundo está cada vez mais dependente de software. No entanto, o conhecimento embutido no software é pouco ou nada transparente. Acreditamos que uma abordagem intencional da Engenharia de Requisitos apresentada neste trabalho é a melhor maneira para elicitar, modelar e analisar (verificar e validar) requisitos, pois permite o conhecimento sobre o "porquê" de cada requisito.

A seguir analisaremos trabalhos com uma abordagem na transparência de software, pois concordamos com a frase enviada pelo professor **John Mylopoulos** "*Transparency is an interesting quality because it makes it necessary to attach requirements models to software*" [70].

5.2.1.“Uma Análise Inicial sobre como Transparência Software e Confiança se Influenciam Mutuamente” [52].

Neste trabalho seus autores começam explicando como foi feita uma pesquisa na internet na procura por uma definição de transparência, eles acharam que muitas definições de diferentes óticas foram propostas, mas todas coincidiam na seguinte noção, que transparência é aquilo que é o suficientemente aberto para permitir que as coisas sejam profundamente observadas a partir de diferentes perspectivas [52].

5.2.1.1.Confiança como uma Característica da Transparência

É um fato que o software já é o suficientemente pervasivo, onde a internet está conectando indivíduos globalmente e os autores deste trabalho concordam com outros trabalhos antes mencionados [9], [55], [56], [57], [58] onde a transparência parece não ser apenas uma possibilidade remota, mas algo que teremos de entregar mais cedo do que muitos pensaram.

O que este trabalho apresenta é o conceito de confiança como uma das características importante da transparência, embora esta confiança às vezes pode ser enganosa.

Para um cliente a confiança pode ser um componente importante para assegurar a transparência. Segundo [52] a transparência é indispensável, pois permite que se possa melhorar o relacionamento com os clientes já que fornece retroalimentação que visa na melhora do produto. No entanto, o nível de transparência tem que ser gerida para assegurar a confiança.

Este trabalho não faz um detalhamento exaustivo sobre estes dois tópicos, embora ele busque incentivar a procura de uma compreensão mais profunda sobre a forma como a confiança e a transparências podem ser utilizadas em benefício da sociedade.

5.2.2.Transparência e Pureza do Software [54].

Este artigo faz uma introdução sobre software que contém funcionalidades as quais não são anunciadas ou conhecidas pelos usuários e que transtornam ou

distorcem a usabilidade do produto. Embora estas funcionalidades não sejam erros, mas sim operações destinadas pelos desenvolvedores para serem desconhecidas aos usuários finais. O autor exemplifica com o caso dos cavalos de Tróia (*Trojan horses*) e os ovos de Páscoa (*Easter eggs*) que a cada vez são mais comuns e tem se convertido em uma fonte de muitos riscos.

5.2.2.1. Enfoque da Transparência

Meunier define a transparência de software como uma condição para que todas as funções do software sejam divulgadas para os usuários. Além disso, a transparência é necessária para a gestão adequada dos riscos. O termo "transparência" segundo ele deve ser usado em vez de "Plenamente divulgado" ou (*fully disclosed*) para evitar confusão com “a plena divulgação” de vulnerabilidades.

5.2.2.2.Software Puro

Essa liberdade de ser “plenamente divulgado” deve ser nomeada de alguma forma, e o autor propõe o conceito de “Pureza”. Embora software puro teoricamente possa existir sem divulgação, mas a divulgação seria um forte incentivo. Pureza não significa livre de erros ou inalterada desde a sua liberação. É possível que o software puro possua erros ou esteja corrompido. O autor cita um caso relacionado a um serviço de vídeo americano que coletou informações de quem viu uma cena de televisão várias vezes. Essa informação, de que haveria coleta de informações do padrão do uso, foi pouco clara e os usuários ficaram pouco informados sobre essa característica do sistema.

Esse caso ajuda a entender porque a pureza do software é uma propriedade desejável que visa a proteger o usuário de funcionalidades indesejadas do software, tendo este a capacidade de remover estas. É assim que transparência de software e pureza são freqüentemente avaliados, mas não explicitamente identificados. E de fato um software opaco pode ter riscos de segurança para as informações dos usuários além de riscos de negocio a sob a forma de perda da reputação, confiança, vendas e contratos. E é neste contexto que o autor enfatiza que a transparência só não é suficiente, em alguns casos se deve exigir pureza do software como um requisito explicito necessário para diminuir os riscos.

5.2.3. Explorando as Características de *i** que Apóiem a Transparência do Software [53].

Neste trabalho se apresenta que transparência do software deve ser fundamentada em requisitos, e que esta será a base tanto para rastreabilidade ascendente quando para rastreabilidade descendente. Conforme este contexto, os modelos *i** são importantes porque deixam claros os requisitos não-funcionais que impactam a transparência de software.

5.2.3.1. Definindo Transparência de Software

Conforme [53] o software é considerado **transparente,** se torna as informações das quais ele trata também de forma transparente (transparência da informação) e se ele mesmo é transparente, ou seja, ele mesmo informa sobre como ele funciona, o que ele faz e as justificativas do "por que" (a transparência do processo). Os autores abordam a problemática na transparência de software propondo a idéia de **requisitos que sejam legíveis para ambas partes** tanto para os interessados em geral quanto para os desenvolvedores.

A Figura 45 apresenta a proposta da escada da transparência, onde cada um dos níveis nos ilustra que atributos de qualidade devem ser atingidos até à transparência.

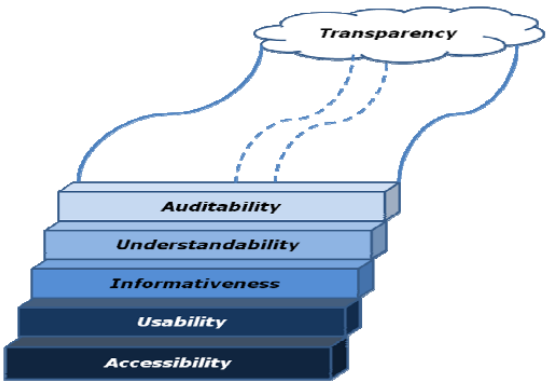

Figura 45 – Escada da Transparência tomada de [53].

5.2.3.2. Como os Modelos *i** Apóiam a Transparência

Nos modelos *i** é possível representar a intencionalidade dos atores, explicitar metas flexíveis, alternativas e intencionalidade detalhada. Ao descrever a **intencionalidade dos atores** estamos atingindo os requisitos não funcionais de rastreabilidade e de verificabilidade que conforme [53] contribuem para a auditabilidade (*auditability*). Do mesmo modo com explicitar **metas flexíveis** nós estamos atingindo os requisitos não funcionais de completeza, clareza e acurácia os quais contribuem para o requisito de ser informativo (*informativeness*). Com a abordagem da **intencionalidade detalhada** se está atingindo os NFRs de decomposição e composição atributos que contribuem para o entendimento (*understandability*) e finalmente a **descrição de alternativas** são importantes para integridade, extensibilidade e validade, os quais estão contribuindo em diferentes níveis da escada da transparência (ver Figura 44).

5.3. Análise dos Trabalhos Relacionados

A seguir apresentamos um resumo das principais características dos trabalhos citados neste capítulo: eles são frutos de um esforço relativamente recente e que aportaram, desde seus diferentes focos, elementos para a elaboração desta dissertação.

Na proposta "Estendendo Tropos para uma Implementação em Prolog: um Estudo de Caso Usando o Problema do Agente Coletor de Alimentos (FCAP)" [49] se faz uso da metodologia Tropos, metodologia está baseada nos requisitos e em no framework *i** para criar os modelos intencionais, tendo similitude com o nosso caso já que a idéia é focar-nos nos requisitos e evoluir os modelos intencionais. Além disso, o trabalho explica que foram usados diagramas de seqüência de cenários para a implementação e no caso do SimulES pretendemos usar o diagrama SDsituations; também para a implementação fizeram uso dos predicados para instanciação de agentes, papéis e posições e nosso caso variou, guiar-nos através do modelo SA para SimulES (Figura 28), além do léxico para a descrição do contexto. Finalmente, os autores fazem uso de cenários como uma boa prática que permite justificar os elementos que estariam presentes tanto na modelagem como na implementação, esta parte é bastante importante, pois o

nosso trabalho também está fortemente baseado em cenários e a nossa idéia é que cenários servirão de mecanismo para mapear modelos até o código.

O trabalho "Tropos: uma Metodologia de Desenvolvimento de Software Orientada a Agentes" [50] também faz uso de Tropos como metodologia de modelagem. Na primeira atividade descrita os atores são identificados e representados no diagrama de atores do contexto *eCulture* (Figura 37) nesta dissertação temos o modelo SA para SimulES (Figura 29) junto com o léxico para esta atividade. Embora este trabalho seja um desenvolvimento para agentes possui itens de interesse para nosso trabalho. Bem que usem o método Tropos identificamos que a modelagem está centrada nos atores. A evolução dos modelos surge com a análise e refinamento dos atores, nesta análise dependências são descobertas e metas são geradas, é assim que isso acontece nas diferentes etapas do Tropos onde novos atores também podem ser descobertos. As dependências identificadas são, segundo os autores aquelas que definirão os requisitos funcionais e não funcionais. Neste trabalho o mapeamento dos modelos até a implementação foi feito usando os diagramas AUML [69] e a ferramenta de desenvolvimento JACK [71], onde planos e metas são programadas para os agentes do sistema.

Continuando com os trabalhos relacionados com a modelagem intencional, a proposta do trabalho "Uma Abordagem para Modelagem Intencional de Avaliação de Riscos de Segurança em *Web Services*" [51] na medida em que se explica como foi feita a modelagem sugere a potencialidade do uso desta, pois acredita que através dela é fácil obter necessidades reais além de fornecer uma análise qualitativa da viabilidade do projeto sob vários cenários. Neste ponto nos acreditamos que uma análise baseada em transparência de software pode avaliar aspectos importantes da implementação que tenha como foco os diferentes usuários. Outro aspecto importante neste trabalho é a modelagem dos ataques ao sistema, isso pode ser levado em consideração em nosso trabalho para representar exceções do sistema (jogo SimulES) ou comportamentos indesejados.

Passando a outro âmbito de nosso trabalho nesta dissertação temos os tópicos relacionados com transparência de software. No primeiro trabalho temos "Uma Análise Inicial sobre como Transparência Software e Confiança se Influenciam mutuamente" [52]. Neste trabalho encontramos novamente a premissa sobre a importância da transparência de software como um requisito que

temos que disponibilizar aos diferentes usuários do software. Aqui é apresentado um conceito novo "Confiança" como um requisito para assegurar a transparência. Se levamos este conceito ao SimulES podemos pensar nele como um atributo não funcional que pode garantir o funcionamento com sucesso do jogo no seu ambiente e durante o tempo que seja determinado. Ao ser um atributo não funcional pode ser qualitativamente definido e analisado além de ser relacionado com outros atributos como custo e desempenho.

A proposta "Transparência e Pureza do Software" [53] analisa as funcionalidades e comportamentos que os usuários esperam que os produtos de software tenham, não obstante, se estes produtos possuem funcionalidades indesejadas criará nos usuários a sensação de incerteza sobre o uso deste. O texto dá ênfase no fato de que as funções do software têm que ser divulgadas para os usuários. Neste contexto acreditamos que a propriedade de "pureza de software" deve aportar ao nosso trabalho em aspetos de divulgação sobre o que o software realmente faz visando a proteção do usuário.

Finalmente no trabalho "Explorando as Características de *i** que Apóiem a Transparência de Software" [54] são apresentados modelos *i** visando representar os requisitos não funcionais que impactam a transparência de software. Este aporte é diretamente ligado a nossa proposta a qual combina modelos intencionais com transparência de software. Além disso, este trabalho enfatiza que os requisitos têm que ser legíveis para que as partes interessadas (interessados em geral e desenvolvedores). Concordamos com essa proposta, já que software deve informar sobre aquilo que faz a seus usuários e desenvolvedores. Se o SimulES é concebido como um produto de software com perspectiva de evolução deve ser o suficientemente transparente. Finalmente o uso de modelos intencionais atingem requisitos não funcionais como auditabilidade, completeza, clareza, acurácia, entendimento, integridade, extensibilidade e validade atributos próprios da transparência que pretendemos explorar neste trabalho.