Pós-Graduação em Ciência da Computação

"ISTAR TOOL - UMA PROPOSTA DE FERRAMENTA PARA MODELAGEM DE I*"

Por

BÁRBARA SIQUEIRA SANTOS

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, AGOSTO/2008

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

BÁRBARA SIQUEIRA SANTOS

"ISTAR TOOL - UMA PROPOSTA DE FERRAMENTA PARA MODELAGEM DE I*"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR(A): Jaelson Freire Brelaz de Castro

RECIFE, AGOSTO/2008

Santos, Bárbara Siqueira

ISTAR TOOL – Uma proposta de ferramenta para modelagem de I* / Bárbara Siqueira Santos. – Recife: O Autor, 2008.

xiii, 249 p. : il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da computação, 2008.

Inclui bibliografia, anexos e apêndices.

1. Engenharia de software. I. Título.

005.1 CDD (22.ed.) MEI2009-015

Dissertação de Mestrado apresentada por **Bárbara Siqueira Santos** Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título, **"IStar Tool - Uma Proposta de Ferramenta para Modelagem de i*"**, orientada pelo **Prof. Jaelson Freire Brelaz de Castro** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Alexandre Marcos Lins de Vasconcelos
Centro de Informática / UFPE

Prof. Oscar Pastor López
Departamento de Sistemas Informáticos e Computação
Universidade Politécnica de Valencia

Prof. Jaelson Freire Brelaz de Castro
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 6 de agosto de 2008.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Para meus pais, Marisete e Romero.

Agradecimentos

Meus sinceros agradecimentos a todos, que direta ou indiretamente, contribuíram para a conclusão deste trabalho e de uma importante fase da minha vida. Aqui começa uma nova etapa com muito entusiasmo, perspectivas, esperanças, projetos e sonhos.

Agradeço a minha família - em especial a minha sobrinha Camila - pelo amor, paciência e apoio a mim dedicados. A Victor pelo amor, companheirismo, paciência, pelas aventuras, sonhos e planos. E aos meus amigos, em especial a Jayrinho e a Jennifer, que sempre estiveram ao meu lado. Com vocês, compartilho mais essa conquista tão importante.

Aos que fazem parte do grupo de pesquisa LER do Centro de Informática, tenho um agradecimento especial pelos seminários, trocas de experiência, momentos de descontração, e acima de tudo pelo suporte acadêmico, técnico e emocional. Em particular, agradeço ao meu orientador o Professor Jaelson Castro pelo apoio e confiança.

Aos meus amigos do SERPRO (Dani, Patrícia, Thaise, André, Davi, Leandro, e Milton) - muito mais que colegas de trabalhos - pelo apoio em uma fase decisiva desta jornada.

Resumo

A orientação a agentes surgiu como um novo paradigma para o desenvolvimento de software e projetistas e desenvolvedores têm se voltado para os conceitos da orientação a agentes a fim de entender, modelar, e desenvolver de forma mais adequada sistemas cada vez mais complexos e que operam em um ambiente distribuído. Como esperado, o crescimento do interesse em agentes de software tem levado a propostas de novas metodologias baseadas nos conceitos de agentes. A metodologia estudada neste trabalho, Tropos, é fundamentada no conceito de agentes. Tais conceitos são adotados desde a fase de requisitos, tal como ator, objetivo e dependência entre atores. A partir do estudo de Tropos, este trabalho focou nas fases de requisitos finais e iniciais, que utiliza i* como linguagem de modelagem. O *framework* i* possui uma estrutura conceitual capaz de reconhecer motivações, intenções e raciocínios sobre as características de um processo, o que facilita os esforços nas atividades da Engenharia de Requisitos. A partir do estudo detalhado do i* e da identificação das restrições sobre o uso desta linguagem, é proposta uma ferramenta - *plug-in* para o Eclipse - com o objetivo de dar suporte ao desenvolvimento dos modelos i* como atividades das fases de requisitos do Tropos. O objetivo deste trabalho é permitir a construção de modelos válidos segundo um guia de boas práticas - i* Wiki, e adotar uma plataforma de desenvolvimento *open source* que possibilite o desenvolvimento colaborativo da ferramenta proposta. Além disto, o *framework* GMF de modelagem gráfica do Eclipse é estudado e adotado como abordagem para a implementação da ferramenta proposta nesta dissertação - IStar Tool.

Palavras-chave: Engenharia de Requisitos, Desenvolvimento Orientado a Agentes, Tropos, i*.

Abstract

Agent orientation has emerged as a new software development paradigm. Software developers are turning to agent-oriented concepts in order to understand, model, and develop software systems which operate within complex and distributed environments. Not surprisingly, the growth of interest in software agents has also led to the proposal of new methodologies based on agent concepts. The methodology studied on this work, Tropos, is one of these proposals and it is founded on agents concepts adopted from Requirements Engineering, such as those of actor, goal and actor dependency. Based on the study of Tropos this work focused on the study of its early and late requirements phases, which relies on i* modeling language. The i* framework provides a conceptual structure able to recognize motivations, intentions and reasoning about the characteristics over an environment (process). This facilitates the effort in the Requirements Engineering phases. The detailed study about i* leads us to a proposal of a tool (an Eclipse Plug-in) in order to provide a suitable supporting on the development of i* models as activities of the early and late requirements phases of Tropos. The goal of this work is to allow building of valid i* models according to a i* guide - i* Wiki, and to adopt a open source platform of development that can permit the collaborative development of the tool proposed. To develop the proposed tool - IStar Tool, the Graphical Modelling Framework (GMF) for the development of plug-in Eclipse is studied and adopted.

Keywords: Requirements Engineering, Agent Oriented Development, Tropos, i*.

Conteúdo

Índice de Figuras

Índice de Tabela

Índice de Abreviaturas

ADL	Architecture Description Language
BDI	Beliefs, Desires and Intentions
EMF	Eclipse Modeling Framework
ER	Engenharia de Requisitos
ES	Engenharia de Software
ESOA	Engenharia de Software Orientada a Agentes
FIPA	Foundation for Intelligent Physical Agents
GEF	Graphical Editing Framework
GMF	Graphical Modeling Framework
MaSE	Multiagent Systems Engineering
MDA	Model Driven Architecture
MDD	Model Driven Development
NFR	Non-Functional Requirements
OME	Organizational Modeling Environment
OMG	Object Management Group
RNF	Requisitos Não Funcionais
SD	Strategic Dependency
SR	Strategic Rationale
SMA	Sistema Multi-Agentes
UML	Unified Modeling Language

Capítulo

1 Introdução

Este capítulo apresenta as principais motivações e objetivos para a realização deste trabalho, além de apresentar a estrutura do trabalho.

1.1 Motivação

A Engenharia de Software (ES) para novos domínios de aplicações, tais como gerenciamento de conhecimento, computação *peer-to-peer* ou serviços para Internet, tem lidado com a modelagem e construção de sistemas que devem operar em ambientes complexos capazes de tratar informações distribuídas, dinâmicas e heterogêneas. Com isso, a Engenharia de Software Orientada a Agentes (ESOA) está se tornando uma das áreas mais crescentes na academia e na indústria, fundamentalmente por tratar características inerentes a estes tipos de sistemas, tais como: alta interatividade, não-determinismo, distribuição, adaptabilidade, dentre outras.

A maioria destes sistemas de software existe em ambientes organizacionais e operacionais que estão em constante mudança – onde novos componentes podem ser adicionados, modificados ou removidos a qualquer momento. Isso geralmente ocorre devido a mudanças nas estruturas organizacionais, modelos de negócio, dinâmicas de mercado, estruturas legais e regulamentares, sentimentos públicos e avanços culturais. Contudo, muitos destes sistemas falham no suporte às organizações das quais eles são parte integrante. Uma das causas deste problema é a ausência de um entendimento apropriado da organização pelos projetistas do sistema. Desta forma, devemos projetar metodologias de ES que sejam adaptáveis a mudanças e foquem no esclarecimento e definição dos relacionamentos entre o sistema que se pretende construir e o mundo.

Neste contexto, a Engenharia de Requisitos (ER) vem sendo conhecida como a fase mais crítica do processo de desenvolvimento de sistemas. Isso porque, para construção de um software com qualidade, é preciso que as considerações técnicas estejam em equilíbrio com as sociais e organizacionais desde a modelagem dos sistemas [Bresciani 2004].

Dentre algumas das metodologias encontradas na Engenharia de Requisitos, Tropos se apresenta como uma proposta de metodologia para desenvolvimento de sistemas orientados a agentes, inspirada na análise de requisitos e fundamentada em conceitos sociais e intencionais [Castro 2002][Giorgini 2005a]. As duas características fundamentais da metodologia Tropos são: (i) o uso de conceitos de nível de conhecimento, tais como agente e objetivos, durante todas as fases do desenvolvimento de software; e (ii) o importante papel atribuído à análise de requisitos quando o ambiente e o sistema a ser desenvolvido estão sendo analisados. Tropos tem como objetivo o desenvolvimento de sistemas analisados e modelados de acordo com as reais necessidades de uma organização, buscando um melhor casamento entre o sistema e o ambiente, permitindo uma melhor estruturação dos sistemas. Para tanto, sua abordagem utiliza conceitos de modelagem organizacional.

Os modelos e conceitos utilizados nas fases iniciais do Tropos são oferecidos pelo *framework* i* [Yu 1995] que possui uma estrutura conceitual rica, capaz de reconhecer motivações, intenções e raciocínios sobre as características de um processo, o que facilita os esforços nas atividades da Engenharia de Requisitos. O i* é composto de dois modelos: o modelo de Dependência Estratégica (SD) e o modelo Estratégico da Razão (SR). O modelo SD fornece uma descrição intencional do processo em termos de uma rede de relacionamentos de dependência entre atores relevantes do ambiente. O modelo SR, por sua vez, apresenta uma descrição estratégica do processo, capturando as razões que estão por detrás dos elementos do processo, ou seja, fornece uma análise dos meios para mostrar como os objetivos podem ser cumpridos através das contribuições dos atores. Tanto o modelo

SD quanto o modelo SR do i* são usados nas fases iniciais do Tropos para capturar as intenções dos *stakeholders*[1], as responsabilidades do novo sistema em relação a estes *stakeholders*, a arquitetura do novo sistema e os detalhes de seu projeto.

Tropos, embora ainda em constante evolução, oferece um *framework* que engloba as principais fases de desenvolvimento de software, com o apoio das seguintes atividades: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural e Projeto Detalhado. Recentemente o escopo de Tropos foi estendido passando a incluir técnicas para geração de uma implementação a partir dos artefatos gerados na fase de Projeto Detalhado. Esta fase é complementada com propostas para plataformas de programação orientada a agentes.

A dissertação, portanto investiga a questão do desenvolvimento de sistemas para o paradigma de orientação a agentes. Em particular, estaremos preocupados em estudar uma proposta de processo para o Tropos focando nas fases de requisitos inicial e final da metodologia e na construção dos seus artefatos de saída - os modelos em i*.

1.2 Objetivos

O principal objetivo deste trabalho é propor uma ferramenta para modelagem i* que permita a construção de modelos válidos segundo o guia do i* [Grau 2008] publicado em [i* Wiki 2008]. A construção de modelos i* é a principal atividade das fases de requisitos inicial e final da metodologia Tropos utilizada pelo grupo de pesquisa LER (Laboratório de Engenharia de Requisitos). Sendo importante, portanto a produção de modelos válidos como artefatos do processo definido nesta metodologia.

Podemos listar como outros objetivos deste trabalho:

- Investigar o paradigma de Engenharia de Software Orientada a Agentes e a proposta da metodologia Tropos;
- Investigar o uso de ferramentas e linguagens de modelagem apropriadas ao desenvolvimento de sistemas multi-agente segundo o Tropos;
- Identificar se as ferramentas estudadas atendem às necessidades do grupo de pesquisa LER - se estão de acordo com o i* segundo [i* Wiki 2008][Grau 2008] e se seguem o processo definido para o Tropos em [Silva, M. J. 2007];
- Criar meta-modelo para o i* segundo [i* Wiki 2008][Grau 2008].

1.3 Estrutura do Trabalho

Este trabalho encontra-se estruturado da seguinte forma:

[1] *Stakeholders* são pessoas ou organizações que serão afetadas pelo sistema e que possuem influência, direta ou indireta, sobre os requisitos do sistema [Kotonya e Sommerville 1998]. Eles incluem usuários finais do sistema, gerentes e outros envolvidos no processo da organização influenciada pelo sistema, clientes da organização que usarão o sistema para obter algum serviço, etc.

1. O capítulo 2 apresenta os conceitos básicos e as características envolvidas nesse trabalho como: o uso da abordagem de agentes, metodologias propostas para o desenvolvimento de sistemas orientados a agentes, padrões e plataformas para desenvolvimento de agentes.
2. O capítulo 3 apresenta os principais aspectos envolvidos no desenvolvimento de sistemas baseado na metodologia Tropos, seguindo suas diversas etapas. Além de apresentar ferramentas que se propõem a dar suporte à metodologia e/ou à criação de modelos i*. Ao final deste capítulo é realizada uma análise entre as ferramentas i* existentes.
3. O capítulo 4 mostra os aspectos envolvidos na proposta do meta-modelo para o i* de acordo com [i* Wiki 2008][Grau 2008]. Além disso, é apresentado um *framework* para o desenvolvimento da ferramenta que é orientado a modelos e está de acordo com a OMG [MDA 2008], e é apresenta a implementação da ferramenta segundo este *framework*.
4. O capítulo 5 apresenta exemplos de uso da ferramenta proposta neste trabalho - utilização de alguns cenários do guia do i* para verificar o comportamento da ferramenta com implementação descrita no capítulo 4.
5. O capítulo 6 apresenta as conclusões finais a cerca do trabalho apresentado e considerações para o desenvolvimento de trabalhos futuros.
6. As referências bibliográficas utilizadas para produção deste trabalho são apresentadas.
7. O apêndice A apresenta uma comparação entre o meta-modelo para o i* criado para a ferramenta proposta nesta dissertação, e os meta-modelos já publicados e relacionados em [Ayala 2006][Lucena 2008].
8. E por fim, se encontram os anexos, referências importantes utilizadas no desenvolvimento dessa dissertação:
    a. Anexo A - Catálogo com os erros mais freqüentes no uso do i*.
    b. Anexo B - Guia da linguagem de modelagem i* usado como referência deste trabalho [Grau 2008].

Capítulo

2 Engenharia de Software Orientada a Agentes

A Engenharia de Software tem lidado com a construção de sistemas que operam em ambientes cada vez mais complexos, distribuídos e dinâmicos. Neste contexto, a orientação a agentes aparece como uma abordagem bastante apropriada para construção de tais sistemas. Isso se deve, fundamentalmente, por tratar-se de uma abordagem que provê um nível de abstração capaz de envolver as características e os comportamentos que estes sistemas apresentam. Sendo assim, metodologias de desenvolvimento têm sido propostas para que tais sistemas sejam construídos de forma mais adequada. Portanto, este capítulo discute o estado da arte da Engenharia de Software Orientada a Agentes (ESOA) e apresenta algumas metodologias existentes.

2.1 Introdução

Com o crescimento da complexidade, distribuição e tamanho das aplicações industriais, a Engenharia de Software (ES) tem procurado obter um melhor entendimento das características dos sistemas, e dessa forma tem estudado diferentes abordagens para atender à necessidade de um novo meio de analisar, projetar e construir softwares. Nesse contexto, a Engenharia de Software Orientada a Agentes (ESOA) se apresenta como uma nova área que surgiu em resposta a esta necessidade e que contempla aspectos importantes de sistemas que envolvem complexidade e distribuição: a interação, a descentralização da informação, e múltiplas perspectivas e métodos de resolução.

A ESOA se preocupa com metodologias e técnicas para produção de softwares orientados a agentes com qualidade. Segundo [Jennings e Wooldridge 2001] o uso da abordagem orientada a agentes é justificado pelos seguintes argumentos:

(i) A abordagem conceitual é adequada para construir soluções de software para sistemas complexos e;

(ii) As abordagens orientadas a agentes representam um verdadeiro avanço sobre o estado da arte na engenharia de sistemas complexos.

A aplicação dessa tecnologia no desenvolvimento de sistemas segundo [Schwambach 2004] se justifica quando observadas algumas características no ambiente em que o sistema irá atuar, tais como:

(i) O domínio da aplicação envolve distribuição de dados, capacidade de resolução de problemas e responsabilidades;

(ii) Há necessidade de se manter autonomia das partes envolvidas, bem como a estrutura organizacional;

(iii) Há interações incluindo negociação, compartilhamento de informações e coordenação;

(iv) O ambiente do domínio da aplicação apresenta características como não-determinismo – a solução do problema não pode ser inteiramente descrita do início ao fim, pois pode haver mudanças no ambiente, imprevisibilidade e dinamicidade.

Justifica-se ainda o uso da abordagem de agentes pelo fato desta apresentar técnicas poderosas que permitem o melhor gerenciamento da complexidade dos sistemas em desenvolvimento, como por exemplo [Jennings e Wooldridge 2001]:

(i) A aplicação de decomposições é uma maneira eficaz de repartir o espaço do problema de um sistema complexo;

(ii) As abstrações presentes na abordagem orientada a agentes são um meio natural de modelar sistemas complexos;

(iii) A abordagem é apropriada para lidar (identificar e gerenciar) com os relacionamentos organizacionais (dependências e interações) que existem em um sistema complexo.

Nas subseções seguintes, serão apresentados os principais conceitos relativos ao paradigma de agentes e onde esta abordagem pode ser utilizada na indústria.

2.1.1 Conceito de Agentes

Referente ao conceito de agentes há um grande debate a cerca do que exatamente constitui um agente – não se chegando ainda a um consenso geral. Por exemplo, segundo [Wooldridge 1997], *agente é um sistema computacional encapsulado que está situado em algum ambiente e é capaz agir de forma flexível e autônoma neste ambiente, a fim de atingir os objetivos para os quais foi projetado*.

De acordo com a [FIPA 2007], organização de padronização para sistemas de agentes, agente é: *uma entidade que reside em um ambiente onde interpreta os dados através de sensores que refletem eventos no ambiente e executam ações que produzem efeitos no mesmo* – podendo o agente ser um software ou hardware puro.

Agentes podem ser úteis como entidades isoladas às quais são delegadas tarefas particulares, ou estes podem existir em ambientes contendo outros agentes. Neste último caso, eles têm de cooperar, negociar e coordenar ações [Zambonelli 2003].

Em um ambiente contendo vários agentes, o conceito de organização é bastante empregado e inclui: agentes, ambiente, organização, papéis, objetivos, responsabilidades e interações. Observe que quando tratamos desse conjunto de agentes autônomos, que interagem entre si e com o ambiente, estando imersos e atuando neste, utilizamos o conceito de Sistemas Multi-Agentes (SMA).

Agentes possuem as seguintes propriedades, dentre outras [Jennings 2000c]:

(i) Autonomia – habilidade do software de agir independentemente sem intervenção direta humana ou de outros agentes.

(ii) Proatividade – agentes não agem simplesmente em resposta ao seu ambiente, eles são capazes de exibir proatividade, ter um comportamento dirigido a objetivo e tomar a iniciativa quando necessário.

(iii) Sociabilidade – habilidade de participar de muitos relacionamentos, interagindo com outros agentes ao mesmo tempo ou em tempos diferentes.

Em se tratando de SMA cada agente pode desempenhar um ou mais papéis, possuir um conjunto bem definido de responsabilidades e visar atingir objetivos. Esses objetivos podem ser comuns a todos os agentes ou não. Para atingir seus objetivos, os agentes interagem uns com os outros, sendo essa comunicação suportada por uma linguagem de comunicação e por ontologia que pode ser usada para associar significado ao conteúdo das mensagens [Schwambach 2004] .

2.1.2 Utilizando Agentes como Abordagem

Nos últimos anos, várias empresas se interessaram em desenvolver aplicações usando a tecnologia de agente, aplicando-a em diferentes domínios. Esta tecnologia está sendo utilizada em aplicações industriais, comerciais e de entretenimento. Algumas das áreas onde abordagens orientadas a agentes estão sendo utilizadas são:

(i) Telecomunicações: controle de rede, transmissão e chaveamento, gerência de serviço e gerência de rede [Luck 2003];

(ii) Controle de Tráfego Aéreo: Agentes estão sendo usados para representar tanto sistemas de controle de tráfego aéreo em operação quanto para representar equipamentos de aviação [Rudowsky 2004];

(iii) Fabricação: sistemas nessa área incluem projeto de configuração de produtos em fabricação, cronograma para fabricação de produtos, controle das operações de fabricação, e determinação de seqüências de produção para uma fábrica [Silva, I. 2005];

(iv) Gerência de informação: Mecanismos de busca são realizados por agentes, que agem autonomamente para buscar na web em benefício de algum usuário. De fato esta é provavelmente uma das áreas mais ativas para aplicações de agente. Como exemplos temos: um assistente pessoal que aprende sobre os interesses do usuário e com base neles compila um jornal pessoal, um agente assistente para automatizar várias tarefas de usuário em um desktop de computador [Silva, I. 2005];

(v) Comércio eletrônico: Algumas tomadas de decisão comerciais podem ser colocadas nas mãos de agentes. Um aumento da quantidade de comércio está sendo empreendido por agentes. Entre estas aplicações comerciais encontra-se um agente que descobre os produtos mais baratos, um assistente pessoal de compras capaz de buscar lojas on-line para disponibilizar o produto e informação sobre o preço, um mercado virtual para comércio eletrônico e vários catálogos interativos baseados em agentes [Silva, I. 2005];

(vi) Gerência de processo de negócio: Organizações têm procurado desenvolver sistemas de Tecnologia da Informação para dar assistência a vários aspectos da gerência de processos de negócio. Agentes podem ser empregados em sistemas de *workflow* (i.e., sistemas que visam garantir que as tarefas necessárias serão feitas pelas pessoas certas nas horas certas) [Rudowsky 2004].

(vii) Agentes têm um papel óbvio nos jogos de computador, cinema interativo, e aplicações de realidade virtual: tais sistemas tendem a ser cheios de caracteres animados autônomos, que podem naturalmente ser implementados como agentes [Silva, I. 2005].

Diante das possibilidades e da necessidade de uso crescente dessa nova abordagem, alguns desafios e obstáculos foram encontrados. Deste modo, torna-se importante o uso de notações, ferramentas e metodologias específicas para o desenvolvimento de software orientado a agentes, visando a construção de sistemas mais robustos, confiáveis, reutilizáveis e de qualidade.

2.2 Metodologias de Desenvolvimento Orientado a Agentes

A construção de sistemas orientados a agentes engloba todos os problemas envolvidos no desenvolvimento de sistemas distribuídos e concorrentes tradicionais, e ainda as dificuldades adicionais que surgem dos requisitos de flexibilidade e interações sofisticadas. Neste contexto, as metodologias para ESOA devem prover abstrações e notações adequadas, além de ferramentas para modelar tanto as tarefas individuais dos agentes quanto as tarefas sociais, ou organizacionais.

Nos últimos anos várias metodologias têm sido propostas para dar suporte à construção de sistemas orientados a agentes, nas subseções seguintes apresentaremos as metodologias mais divulgadas e utilizadas hoje.

2.2.1 GAIA

GAIA [Wooldridge 2000] representa uma tentativa de obter uma metodologia completa e genérica para especificar, analisar e projetar sistemas orientados a agentes. Nesta metodologia, SMAs são visualizados como sendo um conjunto composto de um grande número de entidades autônomas (como uma sociedade organizada por indivíduos) que apresentam um ou mais papéis específicos. A metodologia GAIA é aplicável a um grande conjunto de SMA, além de tratar aspectos de nível macro (sociedade) e micro (agente) dos sistemas [Zambonelli 2003]. GAIA é baseada na visão de que um sistema multi-agente se comporta como uma organização computacional que consiste de vários papéis interagindo a fim de realizar os objetivos do sistema. Essa visão permite que um analista vá sistematicamente estabelecendo os requisitos até chegar a um projeto que seja suficientemente detalhado a ponto de ser implementado [Wooldridge 2000].

A metodologia GAIA propõe uma abordagem onde, a análise e o projeto do sistema podem ser vistos como um processo de desenvolvimento evolutivo de modelos detalhados do sistema a ser construído. Os modelos usados na metodologia estão ilustrados na *Figura 1* de acordo com [Zambonelli 2005], e agrupados por fase: Análise, Projeto Arquitetural e Projeto Detalhado.

O processo GAIA inicia com a fase de análise, cujo objetivo é coletar e organizar as especificações para o projeto da organização [Zambonelli 2005]. O objetivo da fase de análise é desenvolver um entendimento do sistema e suas estruturas - sem referência a qualquer detalhe de implementação. Este entendimento é capturado através do estudo da organização do sistema, que é vista como uma coleção de papéis que tem relacionamentos (interação) com outros papéis.

Um papel pode ser visto como uma descrição abstrata da funcionalidade esperada de uma entidade. Um papel é definido pelos seguintes atributos, a serem utilizados no modelo de papéis [Wooldridge 2000]:

(i) Protocolos: define a maneira de interação com os outros papéis.

(ii) Permissões: são os direitos associados a um papel, indicando os recursos disponíveis para que o papel possa ser desempenhado.

(iii) Atividades: as Atividades de um papel definem as tarefas por ele executadas sem a interação com outros agentes. Podem ser descritas como um conjunto de ações “privadas”.

(iv) Responsabilidades: determinam a funcionalidade e o atributo chave de um papel. As Responsabilidades estão subdivididas em responsabilidades de progresso (do inglês *liveness*) e de segurança (do inglês *safety*) e são descritas na forma de expressões.

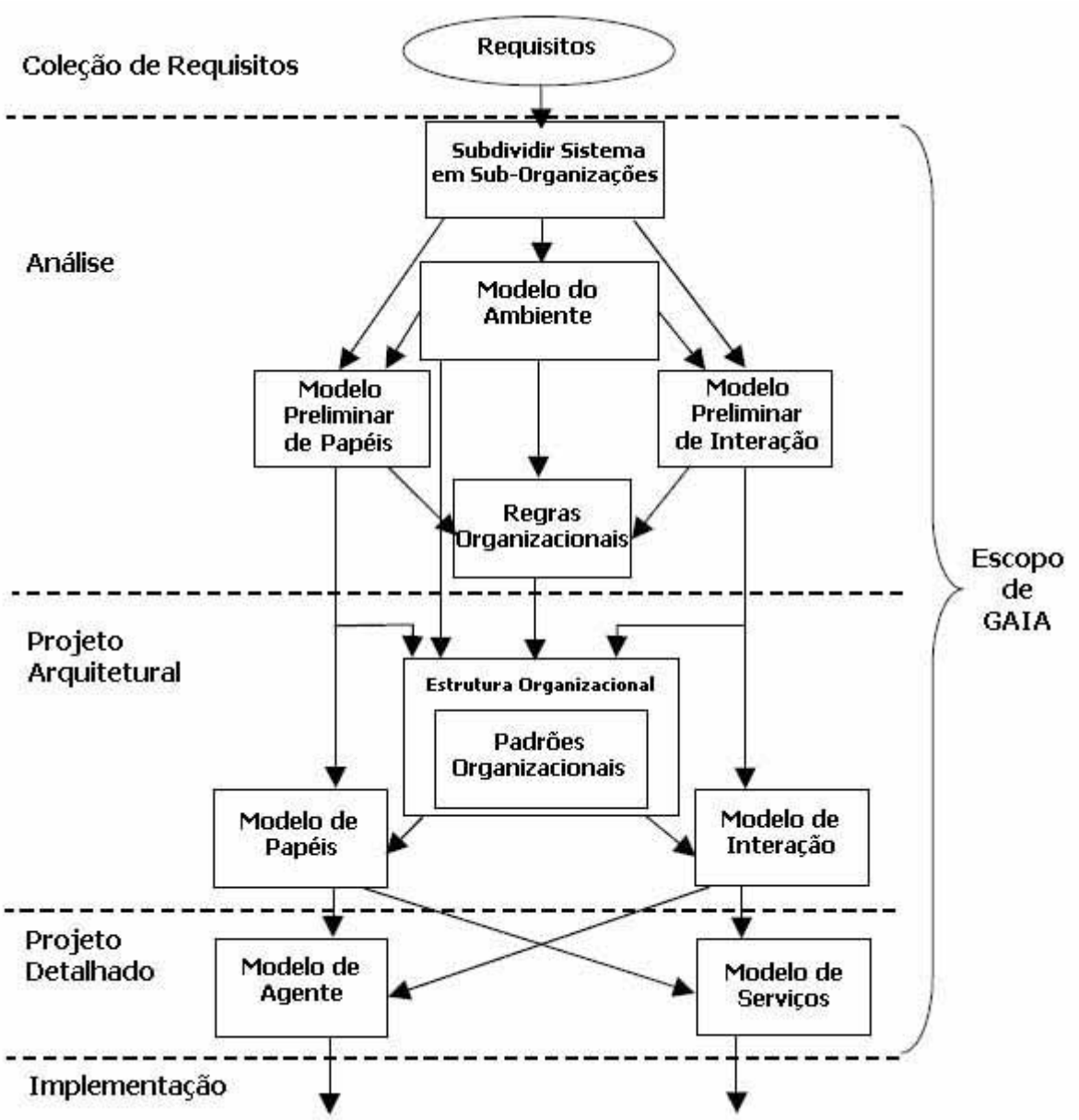


Figura 1 - Modelo da metodologia GAIA e suas relações no processo GAIA [Zambonelli 2005].

A fase de análise, como ilustrado na *Figura 1* inclui a identificação de:

(i) Objetivos das organizações que constituem o sistema como um todo e seu comportamento global esperado. Isto envolve a identificação de como decompor a organização global em sub-organizações.

(ii) Modelo do Ambiente. O ambiente é representado de forma abstrata, uma representação computacional do ambiente no qual o SMA estará situado.

(iii) Modelo Preliminar de Papéis. Identifica as características básicas exigidas pela organização. Este modelo preliminar contém apenas aqueles papéis, possivelmente não completamente definidos, que podem ser identificados sem estarem comprometidos com uma estrutura organizacional específica.

(iv) Modelo Preliminar de Interação. Identifica as interações básicas necessárias para atender as atividades dos papéis preliminares. Este modelo não está comprometido com uma estrutura organizacional específica e, portanto pode estar incompleto.

(v) As regras que a organização deve respeitar. Tais regras expressam restrições relativas à execução de atividades dos papéis e protocolos, e são muito importantes para se obter sucesso no projeto do SMA.

A saída da fase de análise - que consiste de um modelo do ambiente, um modelo preliminar de papéis, um modelo preliminar de interações, e um conjunto de regras organizacionais - é explorada pela fase de projeto, que pode ser decomposta em fase de projeto arquitetural e fase de projeto detalhado [vide *Figura 1*]. A fase de projeto arquitetural inclui [Zambonelli 2005]:

(i) A definição da estrutura organizacional do sistema em termos de sua topologia e regime de controle. Esta atividade, que também poderia explorar catálogos de padrões sociais, considera: (a) a eficiência organizacional, (b) a organização do mundo real (se houver) na qual o SMA estará situado, e (c) a necessidade de cumprir as regras organizacionais.

(ii) Completar os modelos preliminares de papel e de interação. Isto é baseado na estrutura organizacional adotada e envolve - quando possível - os aspectos independentes da organização (detectados na fase de análise) e os aspectos dependentes da organização (derivados da adoção de uma estrutura organizacional específica).

Uma vez que a arquitetura do sistema é identificada juntamente com os modelos de papel e interação completos, a fase de projeto detalhado pode ser iniciada. Esta fase cobre [Zambonelli 2005]:

(i) A definição do modelo de agente. Este modelo identifica as classes de agente que farão parte do sistema.

(ii) A definição do modelo de serviços. Este modelo identifica os principais serviços - blocos de atividades nos quais os agentes se engajam - que são necessários para a realização dos papéis dos agentes, e suas propriedades.

É importante deixar claro o escopo e as limitações da metodologia GAIA:

(i) GAIA não lida diretamente com uma técnica de modelagem particular. Ela propõe, mas não exige técnicas específicas para modelagem (por exemplo, de papéis, de ambiente, de interações).

(ii) GAIA não lida diretamente com caracterísiticas de implementação.

(iii) GAIA não lida explicitamente com as atividades de elicitação e modelagem de requisitos.

2.2.2 Agent UML (AUML)

Agent UML (AUML - *Agent Unified Modeling Language*) [Odell 2001] não se trata de uma metodologia, e sim de uma linguagem de modelagem proposta que visa acomodar as características de agentes na análise e no projeto de sistemas estendendo a já conhecida, padronizada e bastante utilizada UML. AUML tem o objetivo de ajustar o uso de UML - que hoje é amplamente usado no

desenvolvimento de software orientado a objetos - para o seu uso no desenvolvimento de software orientado a agentes. Agentes são vistos como objetos ativos, exibindo autonomia dinâmica (capacidade de ação pró-ativa) e autonomia determinística (autonomia para recusar uma solicitação externa). O objetivo da AUML é fornecer uma semântica semi-formal e intuitiva, através de uma notação gráfica amigável para o desenvolvimento de sistemas orientados a agentes.

A AUML fornece extensões da UML, adicionando uma representação em três camadas para os protocolos de interação de agentes, que descrevem um padrão de comunicação com uma seqüência permitida de mensagens entre agentes e as restrições sobre o conteúdo destas mensagens. Os protocolos de agentes, representados pela abordagem em camadas, definem agora: modelos e pacotes; diagramas de seqüência e colaboração; e máquinas de estados e diagramas de atividades.

2.2.3 MaSE (Multiagent Systems Engineering)

MaSE [DeLoach 2006] foi originalmente projetada para desenvolver sistemas multi-agente de propósito geral e tem sido usada para projetar sistemas robóticos cooperativos. Ela suporta duas fases do ciclo de vida de software (*Figura 2*): Análise e Projeto.

O objetivo da fase de análise em MaSE é definir um conjunto de papéis que podem ser usados para atingir os objetivos ao nível de sistema. Este processo é capturado em três passos:

(i) Capturar objetivos extraindo-os dos requisitos: identificar os objetivos e estruturar objetivos construindo uma hierarquia;

(ii) Aplicar casos de uso: os objetivos capturados são traduzidos em casos de uso, representando assim os comportamentos desejados do sistema e a seqüência de eventos;

(iii) Refinar papéis: organizar os papéis gerando um modelo de papéis onde há uma descrição de cada papel e a forma de comunicação entre eles.

Na fase de projeto o propósito é transformar papéis e tarefas em uma forma mais intuitiva para implementação – especialmente os agentes e os diálogos entre eles. Esta fase consiste de quatro passos:

(i) Construção de classes de agentes: representa a identificação das classes de agentes e seus diálogos e sua representação em Diagramas de Classes de Agentes. Os diagramas gerados são semelhantes aos diagramas de classes utilizados na orientação a objetos, com duas diferenças básicas: as classes de agentes são definidas por papéis ao contrário de atributos e os relacionamentos entre as classes de agentes são os diálogos[2] ao invés de métodos.

(ii) Construção de diálogos: realização do projeto detalhado de diálogos.

(iii) Agrupar classes de agentes: definição da arquitetura interna de cada agente.

(iv) Projeto do sistema: seleção e documentação da configuração real do sistema e as plataformas onde eles deverão estar distribuídos.

[2] Diálogo: representa o estabelecimento de comunicação com troca de mensagens entre duas classes de agentes exclusivamente [Silva, J. M. C. da 2006].

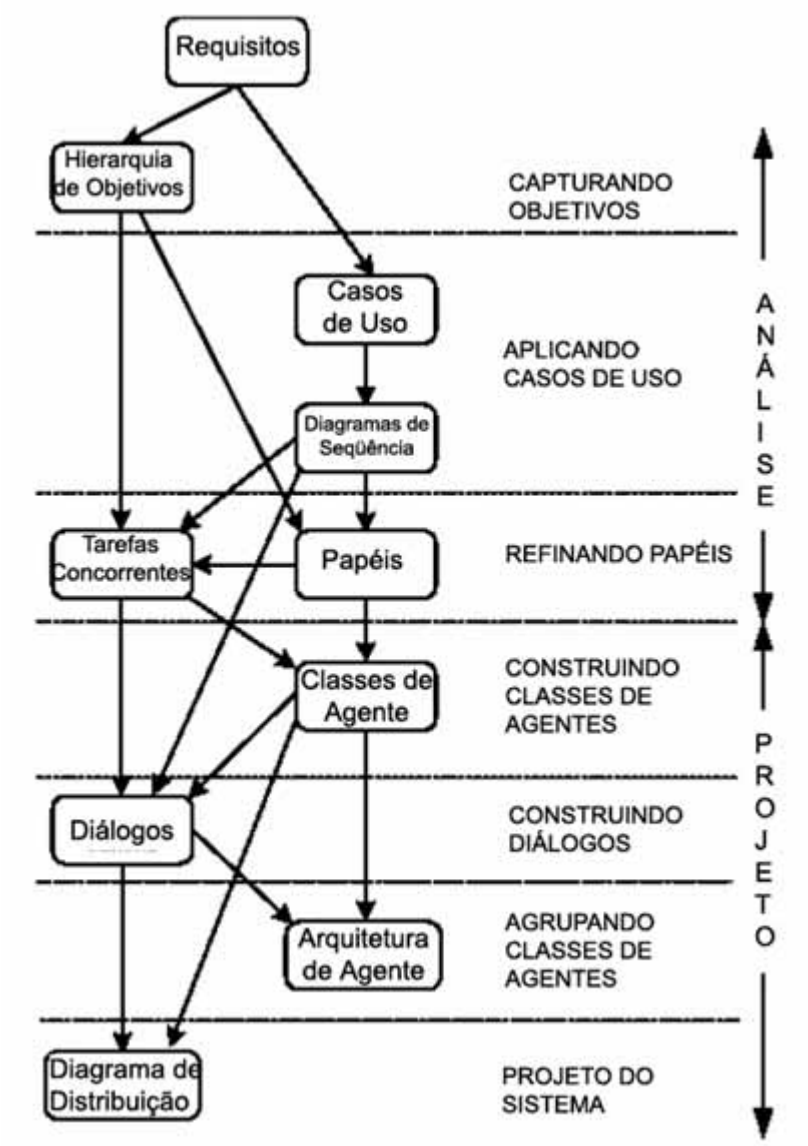


Figura 2 - Fases segundo a metodologia MaSE.

2.2.4 Tropos

Tropos é um projeto que foi lançado em 2000 [Mylopoulos e Castro 2000], e visa apoiar a construção de sistemas de software orientados a agentes. O projeto reúne um grupo de autores de diversas Universidades no Brasil, Canadá, Bélgica, Alemanha, Itália, etc.

A palavra Tropos deriva do grego *tropé*, que significa fácil de mudar ou adaptar. Portanto, Tropos tem como objetivo o desenvolvimento de sistemas de acordo com as reais necessidades de uma organização, buscando um melhor casamento entre o sistema e o ambiente em constante mudança. O processo de desenvolvimento segundo esta metodologia inicia com um estudo e elaboração de um modelo do ambiente no qual o sistema em desenvolvimento irá operar. Este modelo é refinado até que este represente o ambiente com o sistema em seu contexto. Cada modelo é descrito em termos dos atores observados no ambiente em modelagem, seus objetivos e relacionamentos. Tais modelos são gerados segundo os conceitos oferecidos pelo *framework* i* proposto por [Yu 1995][Yu 2001a][Yu

2001b][Yu 2002][Yu e Liu 2005]. Este *framework* possui uma estrutura conceitual rica, capaz de inserir no processo de desenvolvimento de software conceitos intencionais e sociais.

O *framework* é formado por dois modelos:

(i) Modelo de Dependência Estratégica (SD, *Strategic Dependency model*): fornece uma descrição intencional do processo em termos de uma rede de relacionamentos de dependência entre atores relevantes.

(ii) Modelo Estratégico da Razão (SR, *Strategic Rationale model*): apresenta uma descrição estratégica do processo, em termos de elementos do processo e das razões que estão por detrás deles, ou seja, fornece uma análise meio-fins de como os objetivos podem ser atingidos através das contribuições dos demais atores.

Tanto o modelo SD quanto o modelo SR do i* são usados por Tropos para capturar as intenções dos *stakeholders*, as responsabilidades do novo sistema em relação a estes *stakeholders*, a arquitetura do novo sistema e os detalhes de seu projeto.

Embora ainda em constante evolução, Tropos oferece um *framework* que engloba as principais fases de desenvolvimento de software, com o apoio das seguintes atividades: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural e Projeto Detalhado. Recentemente seu escopo foi estendido passando a incluir técnicas para geração de uma implementação a partir dos artefatos gerados na fase de Projeto Detalhado. Tropos visa, entre outros objetivos, a definição de arquiteturas de software mais flexíveis, robustas e abertas.

Em resumo, o foco de cada fase da metodologia é [Castro 2004][Castro 2005][Giorgini 2005a]:

(i) Requisitos Iniciais (*Early Requirements*): Nesta fase, o objetivo é obter o entendimento do problema estudando as características da organização onde o sistema irá atuar. A saída desta fase é um modelo organizacional (SD e SR) que inclui os atores relevantes e seus respectivos objetivos.

(ii) Requisitos Finais (*Late Requirements*): O objetivo desta fase é descrever o sistema a ser desenvolvido no contexto do ambiente onde ele irá operar. A saída desta fase é uma evolução do modelo organizacional produzido na fase anterior introduzindo no ambiente o sistema.

(iii) Projeto Arquitetural (*Architectural Design*): Nesta fase é definida a arquitetura global do sistema em termos de subsistemas e suas dependências (interconexões através do fluxo de dados e controle). A saída desta fase é a seleção do estilo arquitetural (de acordo com a identificação de papéis) a ser aplicado e o modelo de arquitetura.

(iv) Projeto Detalhado (*Detailed* Design): Para cada componente arquitetural é definido, com mais detalhes, a estrutura (ex.: as entradas, as saídas) e o comportamento (ex.: como ocorre o controle) deste no sistema. Os modelos gerados nesta fase formarão a base para a implementação dos agentes e seus comportamentos (atividades/ações).

Hoje, existem diversas ferramentas que se propõem a suportar a construção dos modelos utilizados em Tropos e até mesmo a suportar a metodologia desde a sua fase de requisitos inicial até a

implementação. No capítulo seguinte daremos destaque ao estudo detalhado da linguagem i* de modelagem, pois ela é utilizada nas fases inicial e final de requisitos do Tropos. Também apresentaremos algumas ferramentas de suporte ao Tropos e a aderência destas aos conceitos da linguagem i* segundo [i* Wiki 2008][Grau 2008].

Na tabela abaixo é apresentada uma análise comparativa do Tropos com as demais metodologias apresentadas: GAIA [Zambonelli 2005] e MaSe [DeLoach 2006]. AUML [Odell 2001] não é colocada nesta análise pois esta não se trata de uma metodologia - como colocado na seção 2.2.2, e sim de linguagem de modelagem proposta e que pode ser adotada para utilização em metodologias orientadas a agentes.

Nesta análise os seguintes fatores foram questionados: (a) a fase de requisitos é contemplada, (b) a fase de análise é contemplada, (c) a fase de projeto é contemplada, (d) os conceitos inerentes à abordagem orientada a agentes são utilizados durante todo o processo de desenvolvimento, (e) utiliza alguma técnica de modelagem particular.

Tabela 1 - Análise comparativa do Tropos com outras metodologias.

	GAIA	**MaSE**	**Tropos**
Fase de Requisitos	Não	Não	Sim
Fase de Análise	Sim	Sim	Sim
Fase de Projeto	Sim	Sim	Sim
Conceitos de agentes nas fases	Análise e Projeto	Análise	Requisitos, Análise e Projeto
Cria um modelo do ambiente	Sim	Não	Sim
Técnica de modelagem particular	Não	Não	i*
Referência	[Zambonelli 2005]	[DeLoach 2006]	[Castro 2004] [Castro 2005] [Giorgini 2005a]

Apresentados os pontos de consideração, dentre as metodologias estudadas tomaremos para nosso estudo a metodologia Tropos. Isso se justifica, pois esta propõe o desenvolvimento orientado a agentes inspirada na análise de requisitos fundamentada em conceitos organizacionais (sociais e intencionais), visando assim minimizar a incompatibilidade entre o sistema gerado e seu ambiente. Tropos utiliza i* para descrever modelos do ambiente, linguagem de modelagem que utiliza conceitos inerentes a abordagem orientada a agentes. Além disso, esta metodologia está sendo estudada pelo grupo de pesquisa LER (Laboratório de Engenharia de Requisitos), o qual é liderado pelo Prof. Jaelson Castro, um dos pré-cursores do Tropos.

2.3 Considerações

Nos últimos anos, várias tentativas foram feitas no sentido de desenvolver técnicas e metodologias de modelagem orientadas a agentes. Entretanto, nenhuma destas técnicas pode ainda ser considerada como uma metodologia completa para a análise e projeto de sistemas orientados a agentes. GAIA, uma das metodologias estudadas, se apresenta como uma metodologia genérica para analisar e projetar sistemas orientados a agentes. Porém, GAIA não lida com as atividades de elicitação e

modelagem de requisitos - fase importante para um entendimento apropriado do que se deseja construir. A linguagem Agent UML, que foi apresentada, visa acomodar as características de agentes nas fases de análise e de projeto de sistemas estendendo a UML. Outra metodologia estudada, MaSE, foi projetada para desenvolver SMA de propósito geral e tem sido usada para projetar sistemas robóticos cooperativos. MaSE, assim como GAIA, suporta duas fases do ciclo de vida de software: Análise e Projeto, não contemplando atividades associadas à fase de requisitos.

Do ponto de vista de metodologias de desenvolvimento de software, é freqüente a apresentação de falha no suporte aos interesses das organizações. Uma das causas deste problema é a ausência de um entendimento apropriado da organização pelos projetistas do sistema, bem como as mudanças que não conseguem ser acomodadas pelos sistemas de software existentes. Assim, devemos projetar metodologias de ES que sejam adaptáveis a mudanças e foquem no esclarecimento e definição dos relacionamentos entre o sistema que se pretende construir e o mundo.

Neste contexto, a Engenharia de Requisitos (ER) vem sendo conhecida como a fase mais crítica do processo de desenvolvimento de sistemas porque, para construção de um software com qualidade, é preciso que as considerações técnicas estejam em equilíbrio com as sociais e organizacionais desde a modelagem de sistemas [Bresciani 2004]. Portanto, nossa dissertação toma para estudo o projeto Tropos e a linguagem i*, que tem como foco a etapa de requisitos.

Capítulo

3 Processo de Desenvolvimento com o Tropos

Este capítulo aborda o desenvolvimento de sistemas segundo a proposta Tropos. Como já foi mencionado no capítulo anterior, esta metodologia oferece um framework que engloba as principais fases de desenvolvimento de software, com o apoio das seguintes atividades: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural e Projeto Detalhado. Nosso estudo nessa metodologia terá o foco no framework para modelagem organizacional i utilizado nas fases de requisitos do Tropos. Em seguida, apresentaremos todas as características das etapas referentes à metodologia Tropos e como os modelos em i* são gerados. Por fim, apresentaremos um estudo das ferramentas existentes que se propõem a dar suporte à modelagem organizacional e ferramentas que se propõe a dar suporte ao Tropos.*

3.1 A Metodologia Tropos

3.1.1 O framework i*

O *framework* i* [Yu 1995][Yu 2001a][Yu 2001b] – onde i* ou i-*star* simboliza *intencionalmente distribuído*, é um *framework* de modelagem conceitual desenvolvido para: a análise de sistemas sob uma visão estratégica e intencional de processos que envolvem vários participantes, ou seja, uma modelagem organizacional.

Para a realização desta análise, os engenheiros de requisitos modelam os *stakeholders* como atores e suas intenções como objetivos. Cada objetivo analisado do ponto de vista do seu ator resulta em um conjunto de dependências entre pares de atores. Desta forma, o ambiente do sistema e o sistema em si são visto como organizações de atores, que dependem da ajuda de outros atores para que seus objetivos sejam atingidos, suas tarefas realizadas e recursos sejam fornecidos.

O *framework* i* é uma abordagem orientada a objetivos utilizada para [Yu 2002]:

(i) Obter uma compreensão mais apurada sobre os relacionamentos da organização, de acordo com os diversos atores do sistema,

(ii) Compreender as razões envolvidas nos processos de decisões, e

(iii) Ilustrar as várias características de modelagem que podem ser apropriadas à Engenharia de Requisitos.

Como já foi mencionado, o i* é formado por dois modelos:

(i) Modelo de Dependência Estratégica (SD, *Strategic Dependency Model*): fornece uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependência entre atores relevantes, ou seja, as relações de dependências externas entre os atores da organização.

(ii) Modelo Estratégico da Razão (SR, *Strategic Rationale Model*): apresenta uma descrição estratégica do processo, em termos de elementos do processo e suas razões, ou seja, fornece uma análise dos meios que indicam como os objetivos podem ser cumpridos através das contribuições dos demais atores.

O *framework* i* foi inicialmente proposto por [Yu 1995], mas hoje existem algumas extensões ou variações para sua versão original. Essas variações surgiram de diferentes grupos de pesquisa para atender ao propósito particular de cada um deles, e com isso surgiram diversas ferramentas de suporte. A existência de diferentes usos do i*, de acordo com [Lucena 2008], pode causar:

(i) Divisão do esforço, ao passo que cada grupo de pesquisa irá focar no desenvolvimento de ferramentas de suporte ao seu próprio i*;

(ii) Erros na semântica entre os projetistas e os leitores de um modelo particular do i*;

(iii) Inibição da adoção/uso do i* por parte de novos usuários.

Assim sendo, para apresentar este *framework* utilizaremos os conceitos básicos definidos em [Yu 1995], mas para apresentar a sintaxe desta utilizaremos como referência a evolução do i* original publicada em [i* Wiki 2008]. Essa evolução de sintaxe da descrição original do i* [Yu 1995] tem ocorrido ao longo da publicação de vários artigos [Yu 2001a][Yu 2001b][Yu 2002][Yu e Liu 2005] referenciados em [i* Wiki 2008].

O i* Wiki, é um projeto criado com a intenção de permitir o trabalho colaborativo a cerca do i *. Ou seja, o objetivo é incentivar o *feedback* e a colaboração de usuários do *framework* a fim de que estes possam sugerir alternativas à sintaxe e à semântica da linguagem utilizada – incluindo possíveis extensões. Além disso, com essa comunidade é possível fornecer aos usuários do i* uma visão comum de sua semântica. Para facilitar este processo de colaboração, o i* Wiki se propõe a hospedar duas versões do guia para o i*, uma versão estável – referência a ser utilizada pelos usuários; e uma versão aberta – acessível para que todos os usuários registrados no i* Wiki possam comentar e fornecer sugestões individuais.

Abaixo apresentamos os conceitos deste *framework*, mas os detalhes específicos do i* serão apresentados no próximo capítulo na construção do seu meta-modelo adicionado às suas restrições. Em nosso estudo a construção deste meta-modelo será útil, pois, de acordo com [Bézivin 2005], através dele podemos especificar os requisitos para desenvolvimento de uma ferramenta de suporte bem como facilitar a extensão de uma linguagem.

Conceitos

De acordo com o *framework* i* [Yu 1995], um ator é considerado uma entidade ativa que: realiza ações para atingir objetivos, exercitando seu *know-how*. O termo ator é utilizado para referenciar genericamente qualquer unidade para a qual dependências intencionais possam ser atribuídas. Atores podem ser considerados: intencionais, por possuírem motivações, intenções e razões por trás de suas ações; e estratégicos, quando não focam apenas no seu objetivo imediato, mas quando se preocupam com as implicações de seu relacionamento com outros atores.

Os atores descritos segundo o *framework* podem ser diferenciados em três especializações (*Figura 3*): papéis, agentes e posições. Os mapeamentos de agente, papel, e posição são úteis para modelagem de um grande número de dependências que agentes (atores) no mundo real geralmente possuem, estes agentes podem executar diversos papéis e participar em vários processos.

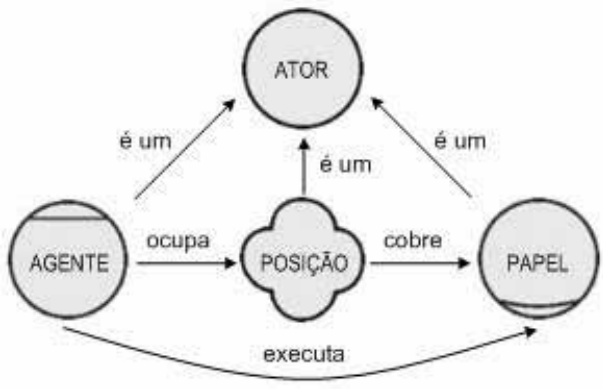


Figura 3 - Atores em i*: representação de agentes, papéis e posições.

A *Figura 3* ilustra a representação gráfica e as possíveis associações entre agentes, papéis e posições. A seguir são definidos os tipos de atores [i* Wiki 2008][Grau 2008]:

(i) Agente é um ator que possui manifestações físicas concretas. Tanto pode referir-se a humanos quanto a agentes de software ou hardware. Um agente pode executar um determinado papel ou ocupar uma determinada posição que cobre um papel.

(ii) Papel representa a caracterização abstrata do comportamento de um ator dentro de determinados contextos sociais ou domínio de informação. Apresenta as funções que podem ser exercidas por um agente dentro da organização.

(iii) Posição representa uma entidade intermediária entre um agente e um papel. É o conjunto de papéis tipicamente ocupados por um agente, ou seja, representa uma posição dentro da organização onde o agente pode desempenhar várias funções (papéis).

As relações entre os atores são descritas através de *links* de associação, como apresentados na *Figura 4*. As associações podem ser de seis tipos [i* Wiki 2008][Grau 2008]:

(i) *Is part of*: Nessa associação cada papel, posição e agente pode ter sub-partes. Em *Is-part-of* há dependências intencionais entre o todo e sua parte. Por exemplo, a dependência do todo sobre suas partes para manter a unidade na organização.

(ii) *ISA*: Essa associação representa uma generalização, com um ator sendo um caso especializado de outro ator. Ambas, *ISA* e *Is-part-of*, podem ser aplicadas entre quaisquer duas instâncias do mesmo tipo de ator.

(iii) *Plays*: A associação *plays* é usada entre um agente e um papel, com um agente executando um papel. A identidade do agente que executa um papel não deverá ter efeito algum nas responsabilidades do papel ao qual está associado, e similarmente, aspectos de um agente deverão permanecer inalterados mesmo associados a um papel que este desempenha.

(iv) *Covers*: A associação *covers* é usada para descrever uma relação entre uma posição e os papéis que esta cobre.

(v) *Occupies*: Esta associação é usada para mostrar que um agente ocupa uma posição, ou seja, o ator executa todos os papéis que são cobertos pela posição que ele ocupa.

(vi) *INS*: Esta associação é usada para representar uma instância específica de uma entidade mais geral. Por exemplo, quando se deseja representar um agente que é uma instanciação de outro agente.

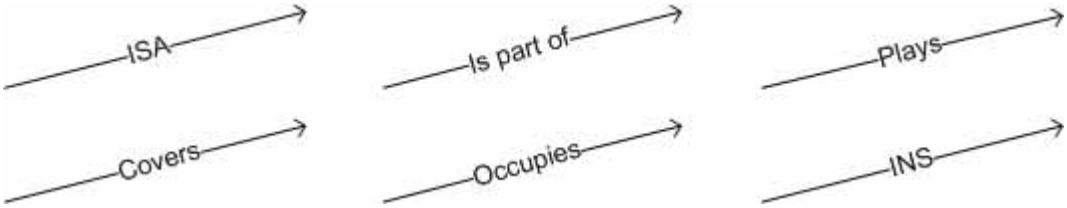


Figura 4 - Associação entre atores.

A seguir detalharemos os modelos já mencionados segundo o *framework* i*, descrevendo os elementos que os compõem. Detalhes relativos às restrições presentes na construção dos modelos serão apresentados no capítulo seguinte durante a definição do meta-modelo do i* segundo [i* Wiki 2008][Grau 2008].

O Modelo de Dependência Estratégica

No modelo SD são capturadas as motivações e os desejos dos atores que fazem parte da organização e apresentamos a sua rede de relacionamentos. Dado um modelo SD, podemos perguntar que novos relacionamentos entre os atores são possíveis, identificar a viabilidade ou não das dependências, ou ainda relacionar os desejos de um ator com as habilidades do ator do qual depende, para poder explorar as oportunidades disponíveis a esses atores.

Cada relacionamento capturado em i* é uma dependência, um acordo (chamado *dependum*), entre dois atores: o *depender* e o *dependee*. O *depender* é o ator que depende de um outro ator (*dependee*) para que o acordo (*dependum*) possa ser realizado. Os relacionamentos de dependência usados em i* descrevem a natureza do acordo e podem ser de quatro tipos: tarefas (*task*), recursos (*resource*), objetivo (*goal*) ou objetivo-*soft* (*softgoal*). Quando o ator *dependee*, numa relação de dependência, disponibilizar um recurso, executar uma tarefa de sua responsabilidade, atender a um objetivo do ator dependente, ou realizar alguma tarefa para alcançar um objetivo-*soft*, teremos a relação de dependência satisfeita pelo *dependee* responsável por ela. A *Figura 5* apresenta os tipos de ligações de dependência no *framework* i*.

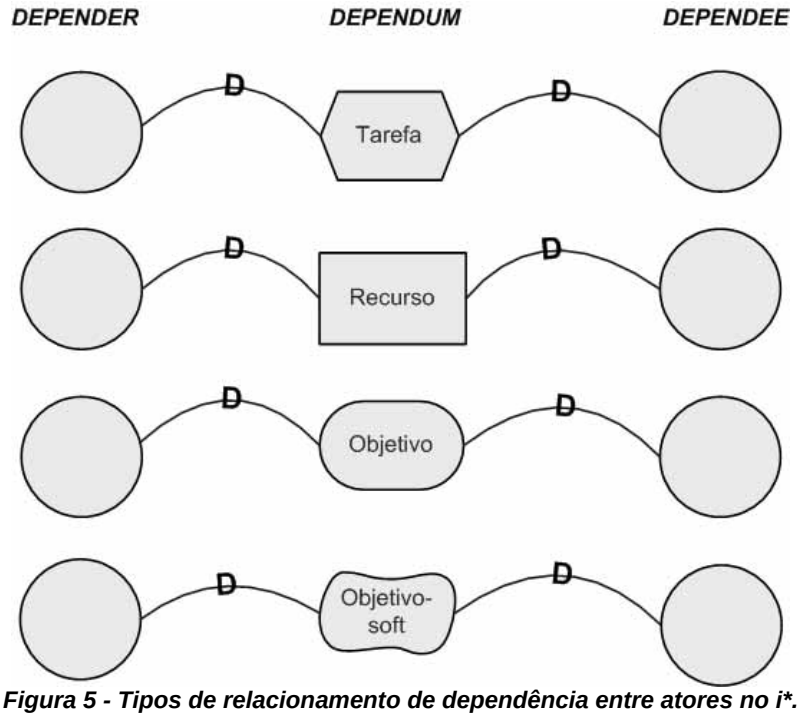


Figura 5 - Tipos de relacionamento de dependência entre atores no i*.

Para cada ligação de dependência representada acima faremos uma descrição de sua utilização:

(i) Na dependência de tarefa, o *dependee* é requisitado a executar uma dada atividade, sendo informado sobre o que deve ser feito. Uma tarefa aponta uma forma particular de se fazer

algo. Tarefas podem ser vistas como soluções que provêem operações, processos, representações de dados, estruturação. Contudo, a descrição de uma tarefa em i* não tem por intenção ser uma completa especificação dos passos necessários à execução da mesma.

(ii) Na dependência de recurso, o ator depende da disponibilidade de uma entidade física ou de uma informação. Por recurso entendemos o produto final de alguma ação, ou de um processo, que estará ou não disponível para o ator dependente. Neste tipo de dependência assume-se que não haja aspectos pendentes a serem tratados ou decisões a serem tomadas.

(iii) Na dependência de objetivo, um ator depende de outro para que uma determinada condição seja satisfeita, não importando a maneira como a satisfação foi alcançada. Por objetivo se entende como uma condição ou estado do mundo que um ator gostaria de alcançar. Geralmente um objetivo é expresso como um desejo.

(iv) A dependência de objetivo-*soft*, de forma similar à dependência de objetivo, também representa um desejo a ser satisfeito, exceto pelo fato de que não há um critério claramente definido a cerca da verificação se a condição desejada foi atingida. A sua satisfação depende do julgamento subjetivo e interpretação dos *stakeholders*.

No modelo SD pode-se identificar ainda diferentes graus de dependências ou vulnerabilidades [i* Wiki 2008][Grau 2008], apresentados graficamente na *Figura 6*:

(i) Aberta (*open*): é usada quando, na falha da relação de dependência, as intenções são afetadas, mas sem conseqüências sérias. A dependência aberta é graficamente representada por “O”.

(ii) Compromissada (*commited*): é usada quando, mesmo com falha na relação de dependência, as intenções não são afetadas. Graficamente não possui sinal relacionado.

(iii) Crítica (*critical*): é usada quando, na falha da relação de dependência, as intenções são afetadas seriamente levando-se a uma preocupação quanto à viabilidade de toda a rede de relacionamentos e dependências. A dependência crítica é graficamente representada por “X”.

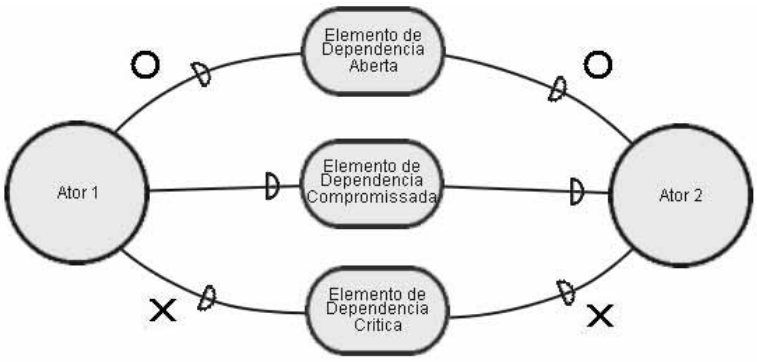


Figura 6 - Graus de dependência (ou vulnerabilidade) em i*.

O Modelo Estratégico da Razão

Enquanto o modelo SD trata apenas dos relacionamentos externos entre os atores, o modelo SR é utilizado para descrever os relacionamentos internos: interesses, preocupações e motivações dos atores participantes de um processo. Ele possibilita a avaliação das possíveis alternativas de definição do processo, investigando mais detalhadamente as razões existentes, ou intencionalidades, por trás das dependências entre os atores. As intencionalidades de cada ator são identificadas e representadas dentro da fronteira de cada ator (*Figura 7*).

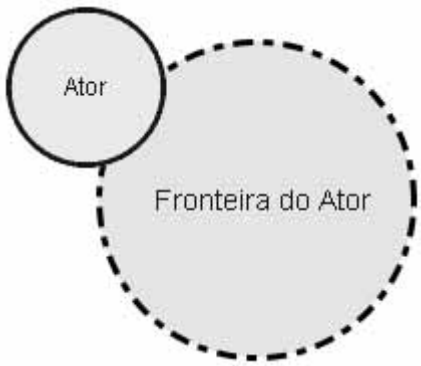


Figura 7 – Modelo SR, fronteira do ator.

Assim como o SD, o modelo SR também é composto por ligações de dependência. Além de utilizar os quatro tipos de dependências apresentados no modelo SD - recurso, tarefa, objetivo e objetivo-*soft*, o modelo SR dispõe de outros dois tipos de relacionamentos: as ligações de meio-fim e as ligações de decomposição de tarefas.

As ligações de meio-fim indicam um relacionamento entre um nó fim e um meio para atingi-lo. Este tipo de ligação sugere alternativas existentes para se alcançar um determinado fim, onde os meios são expressos em forma de tarefas e os fins expressos em forma de objetivos [i* Wiki 2008][Grau 2008]. A representação deste tipo de ligação pode ser visualizada na *Figura 8*.

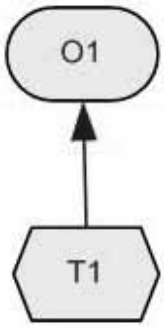


Figura 8 - Ligação meio-fim.

Já as ligações de decomposição de tarefas ligam um nó de tarefa a seus nós componentes, que segundo [Yu 1995] podem ser outras tarefas, objetivos, recursos ou objetivos-*soft*. Uma tarefa decomposta só pode ser considerada completa quando todos os seus nós componentes forem realizados ou satisfeitos. A representação deste tipo de ligação pode ser visualizada na *Figura 9*.

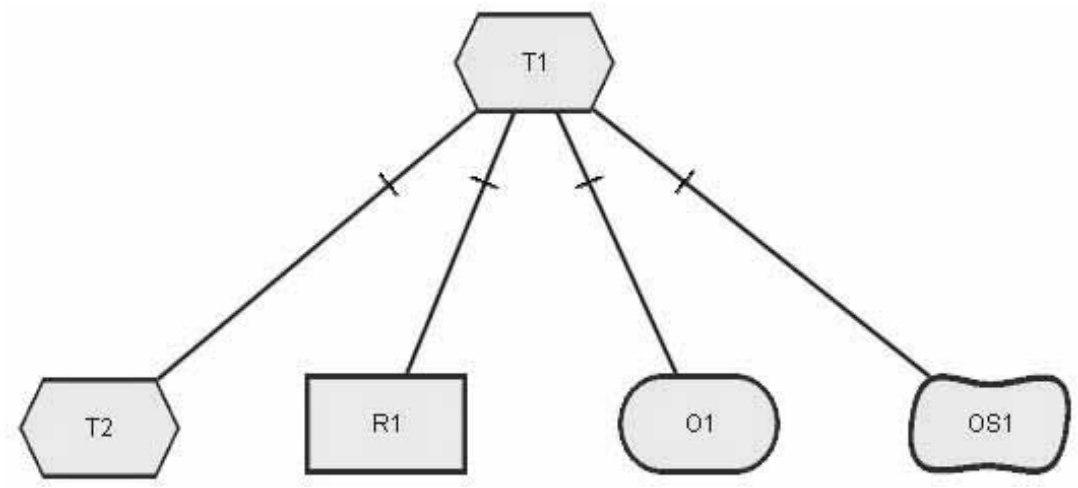


Figura 9 - Elementos de ligação de decomposição de tarefas.

Para construção do modelo SR é realizado um detalhamento de alguns dos atores (ou todos) envolvidos no processo que foram identificados no modelo SD. Neste modelo podemos identificar então o comportamento interno dos atores detalhados, ligações do tipo decomposição de tarefas e ligações do tipo meio-fim.

No modelo SR podemos ainda associar atributos às ligações que envolvem objetivos-*soft*, os *links* de contribuição (*Figura 10*). Quaisquer um dos *links* de contribuição apresentados abaixo pode ser usado para ligar qualquer elemento a um objetivo-*soft*. Essa ligação serve para modelar a forma como os elementos podem contribuir para a satisfação ou cumprimento deste objetivo-*soft*.

Estes atributos indicam o tipo de contribuição relacionada com os objetivos das dependências, podendo ser [Grau 2008]:

(i) *Make*: é uma contribuição positiva, suficientemente forte para satisfazer o objetivo-*soft*.

(ii) *Some* +: é uma contribuição positiva, mas cuja força (influência) é desconhecida.

(iii) *Help*: é uma contribuição positiva parcial, porém não é suficiente para que ela sozinha satisfaça o objetivo-*soft*.

(iv) *Unknown*: é uma contribuição cuja influência é desconhecida.

(v) *Break*: é uma contribuição negativa, suficientemente forte para recusar a satisfação do objetivo-*soft*.

(vi) *Some* -: é uma contribuição negativa, mas a força de sua influência é desconhecida.

(vii) *Hurt:* é uma contribuição negativa parcial, porém não é suficiente para que ela sozinha recuse a satisfação de um objetivo-*soft.*

(viii) *Or:* é uma contribuição cujo destino é satisfeito se algum dos descendentes for satisfeitos.

(ix) *And*: é uma contribuição cujo destino é satisfeito se todos os descendentes forem satisfeitos.

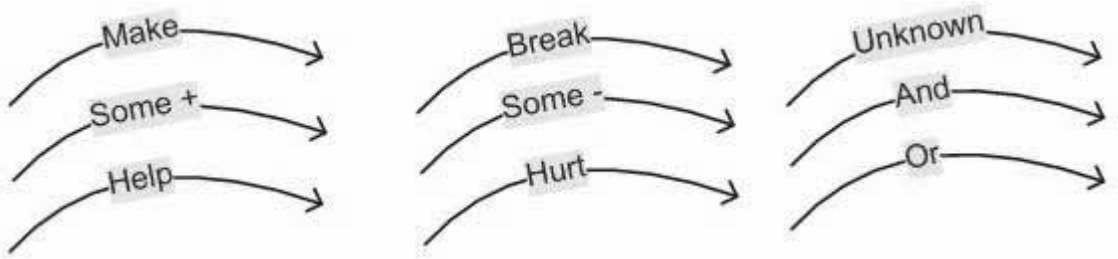


Figura 10 - Links de Contribuição para um objetivo-soft.

3.1.2 Fases do Tropos

Como apresentado na seção anterior, o Tropos utiliza os conceitos e modelos do *framework* i* [Yu 1995] nas fases iniciais do seu processo. Esta seção apresentará resumidamente todas as fases cobertas por esta metodologia e os artefatos produzidos como saída de cada uma delas:

(i) Requisitos Iniciais (*Early Requirements*): Nesta fase, o objetivo é obter o entendimento do problema estudando as características da organização onde o sistema irá atuar.

(ii) Requisitos Finais (*Late Requirements*): O objetivo desta fase é descrever o sistema a ser desenvolvido no contexto do ambiente onde ele irá operar.

(iii) Projeto Arquitetural (*Architectural Design*): Nesta fase é definida a arquitetura global do sistema em termos de subsistemas e suas dependências (interconexões através do fluxo de dados e controle).

(iv) Projeto Detalhado (*Detailed* Design): Para cada componente arquitetural é definido, com mais detalhes, a estrutura (ex.: as entradas, as saídas) e o comportamento (ex.: como ocorre o controle) deste no sistema.

A partir da idéia inicial do projeto Tropos, duas abordagens diferentes surgiram, nomeadas em [Silva, M. J. 2007] de: a versão de Toronto ou Tropos'01 [Castro 2002] e a versão da Itália ou Tropos'04 [Bresciani 2004]. Baseado em [Wautelet 2005], foi proposto em [Silva, M. J. 2007] um processo de desenvolvimento de software em espiral para MAS (*Multi-Agent System*), através da análise dessas duas versões do Tropos. Em particular foi mostrada as diferenças entre as duas versões do Tropos e foram identificadas a seqüência das atividades de cada uma e seus artefatos de saída.

Contudo, foi verificado por [Silva, M. J. 2007] que as atividades de ambas as abordagens são similares nas fases de requisitos iniciais e finais. Nas fases de projeto arquitetural e projeto detalhado foram detectadas algumas diferenças, em geral, as seqüências de atividades produzem alguns artefatos diferentes e outros são iguais.

Baseado neste levantamento das diferenças e semelhanças foi proposta uma seqüência híbrida de atividades. Esta proposta reúne uma seleção de seqüências de atividades que permite aos usuários da metodologia utilizar as melhores características de cada abordagem. A seqüência de atividades selecionadas segundo [Silva, M. J. 2007] é mostrada na tabela abaixo (*Tabela 2*).

Tabela 2 - Seqüência de Atividades do Tropos [Silva, M. J. 2007].

Atividades	Artefatos
Requisitos Iniciais (Early Requirements)	
ER 1 - Identificar os *stakeholders* e suas intenções. ER 2 - Decompor os objetivos através de uma análise orientada a objetivos. ER 3 - Analisar e decompor os elementos intencionais. ER 4 - Identificar os objetivos que contribuem positivamente ou negativamente para os objetivos-*soft*.	A1 - Lista de *stakeholders*. A2 - Modelo de Atores: Diagrama de Atores (SD) e descrição dos elementos. A3 - Modelo de Objetivos: Diagrama de Objetivos (SR) e descrição dos elementos.
Requisitos Finais (Late Requirements)	
LT 1 - Identificar o sistema a ser desenvolvido e os seus relacionamentos com outros atores. LT 2 - Analisar os objetivos do ponto de vista do ator que representa o sistema. LT 3 - Identificar as contribuições dos objetivos-*soft*. LT 4 - Decompor as tarefas. LT 5 - Revisar as dependências no modelo de atores.	A4 - Modelo de Atores: Diagrama de Atores (SD) e descrição dos elementos. A5 - Modelo de Objetivos: Diagrama de Objetivos (SR) e descrição
Projeto Arquitetural (Architectural Design)	
AD 1 - Incluir novos atores e delegação dos seus objetivos-*soft*. AD 2 - Selecionar o estilo arquitetural dentre um conjunto de estilos arquiteturais. AD 3 - Incluir, se necessário, novos atores baseados na seleção do estilo arquitetural. AD 4 - Definir os agentes de acordo com padrões sociais. AD 5 - Definir um conjunto de tipos de agentes (padrões sociais).	A6 - Modelo de atores para a arquitetura do sistema. A7 - Estilo arquitetural selecionado. A8 - Modelo de Atores (SD) - atualizados se novos atores forem adicionados. A9 - Modelo de Objetivos (SR) - atualizados se novos atores forem adicionados. A10 - Lista de capacidades dos atores. A11 - Padrão social selecionado.
Projeto Detalhado (Detailed Design)	
DD 1 - Modelagem do diagrama de classe em UML. DD 2 - Criação do modelo de capacidade. DD 3 - Especificar cada tarefa do diagrama de capacidade através de diagramas de atividade em UML. DD 4 - Modelar os eventos usando diagramas de seqüência e de colaboração em AUML.	A12 - Diagrama de classe em UML. A13 - Diagrama de capacidade. A14 - Diagrama de tarefas. A15 - Diagrama de interação.

Nossa dissertação considerará esta proposta feita por [Silva, M. J. 2007], pois esta reúne as melhores práticas das abordagens existentes para as quatro fases originais do Tropos. Abaixo, as atividades apresentadas na tabela acima, serão apresentadas juntamente com as diretrizes para a fase de implementação. Porém, daremos maior ênfase às fases de requisitos (inicial e final) e aos artefatos

produzidos nestas - os modelos em i*. A nossa preocupação aqui é levantar a importância da construção destes modelos iniciais no Tropos e apontar a falta de padrão a cerca de seu uso.

Utilizaremos como exemplo para demonstrar os artefatos de saída das fases de requisitos do Tropos, a modelagem de um sistema para gerenciamento de conferências. Este exemplo, que vem sendo utilizado pelo grupo de pesquisa na elaboração dos artigos, foi introduzido em [Zambonelli 2003] e modelado usando a metodologia Tropos em [Silva, C. 2006]. Em nosso exemplo, uma conferência envolve diversos atores e três fases. Durante a fase de submissão autores submetem seus artigos e são informados que estes foram recebidos, e recebem um número de submissão. Na fase de revisão, o presidente (*chair*) da conferência deve conduzir a revisão dos artigos que foram submetidos à conferência. Para tanto, ele contata os revisores e os convida para que estes revisem alguns artigos de acordo com sua área de conhecimento. Eventualmente, as revisões irão acontecer e estas serão usadas para decidir sobre a aceitação ou rejeição das submissões. Na fase final os autores precisam ser notificados das decisões. Em caso de aceitação, será requisitada aos autores a produção e submissão da versão revisada de seu artigo. A editora deve então coletar essas versões finais e imprimir os *proceedings* da conferência.

Requisitos Iniciais

Assim como em muitas metodologias de software, a etapa inicial do Tropos é caracterizada pela análise de requisitos. No caso do Tropos a análise e modelagem dos requisitos é realizada seguindo o *framework* i* [Yu 1995] e está dividida em duas fases [Castro 2006]: requisitos iniciais e requisitos finais.

A fase de Requisitos Iniciais está preocupada com o entendimento de um problema estudando-se o ambiente organizacional existente no qual o sistema irá operar. Durante esta fase, os engenheiros de requisitos e analistas de domínio modelam os *stakeholders* como atores, e suas intenções, como objetivos, objetivos-*soft*, tarefas e recursos. Cada intenção é analisada do ponto de vista de seu ator resultando em um conjunto de dependências entre pares de atores.

Além da lista de *stakeholders*, os artefatos de saída desta fase são dois modelos:

(i) Modelo de Atores: Este modelo é equivalente ao modelo de dependência estratégica (SD) que captura os atores relevantes, seus objetivos e suas interdependências;

(ii) Modelo de Objetivos: Este modelo é equivalente ao modelo estratégico da razão (SR) que determina as intencionalidades por trás das relações de dependência.

Tabela 3 - Lista de stakeholders.

Stakeholder	Objetivos
1. Autor e Autor Aceito	Submeter artigo
2. Chair	Realizar a revisão dos artigos; Obter lista dos artigos aceitos; Ter os *proceedings* impressos
3. Revisor	Revisar artigos
4. Editora	Imprimir *proceedings*

Na *Tabela 3* foram listados os *stakeholders* identificados no nosso exemplo da conferência. Os atores identificados foram: o autor (tendo como tipo de ator um autor aceito), o *chair* da conferência, o revisor e a editora. Como veremos nos modelos i* os atores participam da organização colaborando para que os objetivos sejam alcançados. As figuras a seguir exemplificam o uso de modelos em i* nesta fase inicial do Tropos, ilustrando respectivamente o modelo SD (*Figura 11*) e o modelo SR (*Figura 12* e 13) - dividido em duas partes para melhor visualização, a primeira detalhando as intenções do revisor e a segunda a do *chair*.

Os atores que participam da organização, como ilustrado na *Figura 11* são: o Autor, o Autor Aceito (com artigo aceito na conferência), o *Chair* da conferência, o Revisor e a Editora. O *Autor* tem como objetivo *Submeter Artigo*, e ter a sua disposição o *Número de Submissão* do artigo e a *Notificação* de aceitação ou não do artigo. O *Autor Aceito* é um *Autor* que teve seu artigo aceito - relação representada pela associação *ISA* (observar *Figura 11*). O *Chair* da conferência precisa do *Autor Aceito* para ter a sua disposição a *Versão do Artigo Aceito Revisado*. Ele tem como objetivos *Ter Revisão dos Artigos Submetidos*, *Ter Proposta de Revisão Aceita* pelos *Revisores*, *Concordar com Artigos Aceitos* através de discussão com os revisores, e *Obter Proceedings Impressos*. O *Revisor* depende do *Chair* para ter o *Artigo* para realizar a revisão, e a *Editora* depende do *Chair* para ter a *Versão Eletrônica dos Proceedings* para que este seja impresso.

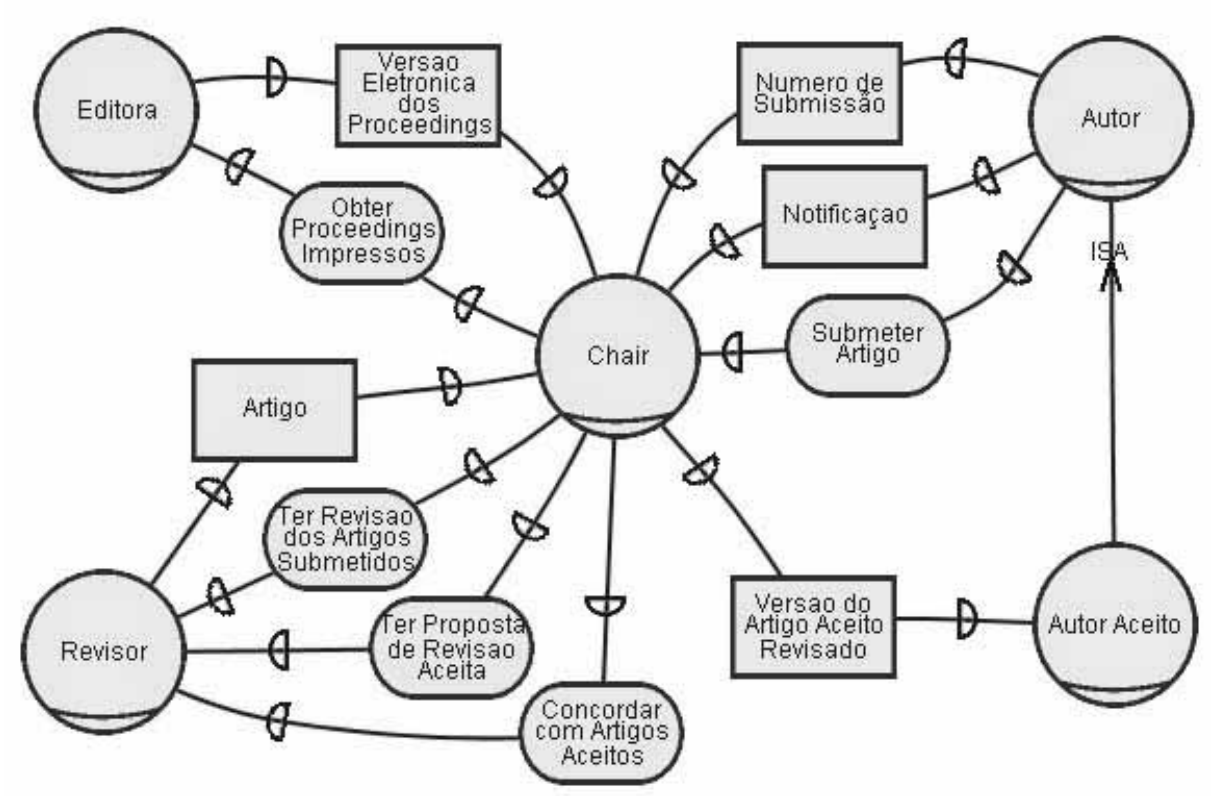


Figura 11 - Exemplo gráfico do SD na fase de Requisitos Iniciais.

No modelo SR temos o detalhamento das intenções dos atores *Revisor* e *Chair*. Como já mencionado, o modelo SR possibilita a análise detalhada das razões existentes, ou intencionalidades, por trás das dependências entre os atores. A *Figura 12* ilustra a análise das intenções do ator *Revisor*.

O principal objetivo do *Revisor* é *Participar de Conferência*. Para atingir este objetivo o ator precisar *Ser Membro do Comitê de Programa*, esta tarefa compreende *Avaliar Proposta para Revisão de Artigo*, *Realizar Revisão de Artigo*, *Discutir Aceitação de Artigo*. O objetivo de *Avaliar Proposta para Revisão de Artigo* pode ser atingido através da tarefa *Avaliar Proposta para Revisão*, que compreende as tarefas *Avaliar Interesse na Área* de estudo referente ao artigo, *Avaliar Relevância da Conferência*, *Avaliar Tempo Disponível* para participar do comitê de programa e realizar revisão, e *Responder Proposta* para revisar artigos.

A tarefa *Realizar Revisão de Artigo* auxilia o ator *Chair* a atingir seu objetivo de *Ter Revisão dos Artigos Submetidos*, e a tarefa *Discutir Aceitação de Artigo* auxilia a atingir o objetivo de *Concordar com Artigos Aceitos*. *Realizar Revisão de Artigo*, do ator *Revisor*, é uma tarefa que compreende as tarefas *Preencher Formulário de Revisão* e *Avaliar Artigo*. Para executar esta última tarefa o *Revisor* precisa que o *Chair* disponibilize o *Artigo* para revisão.

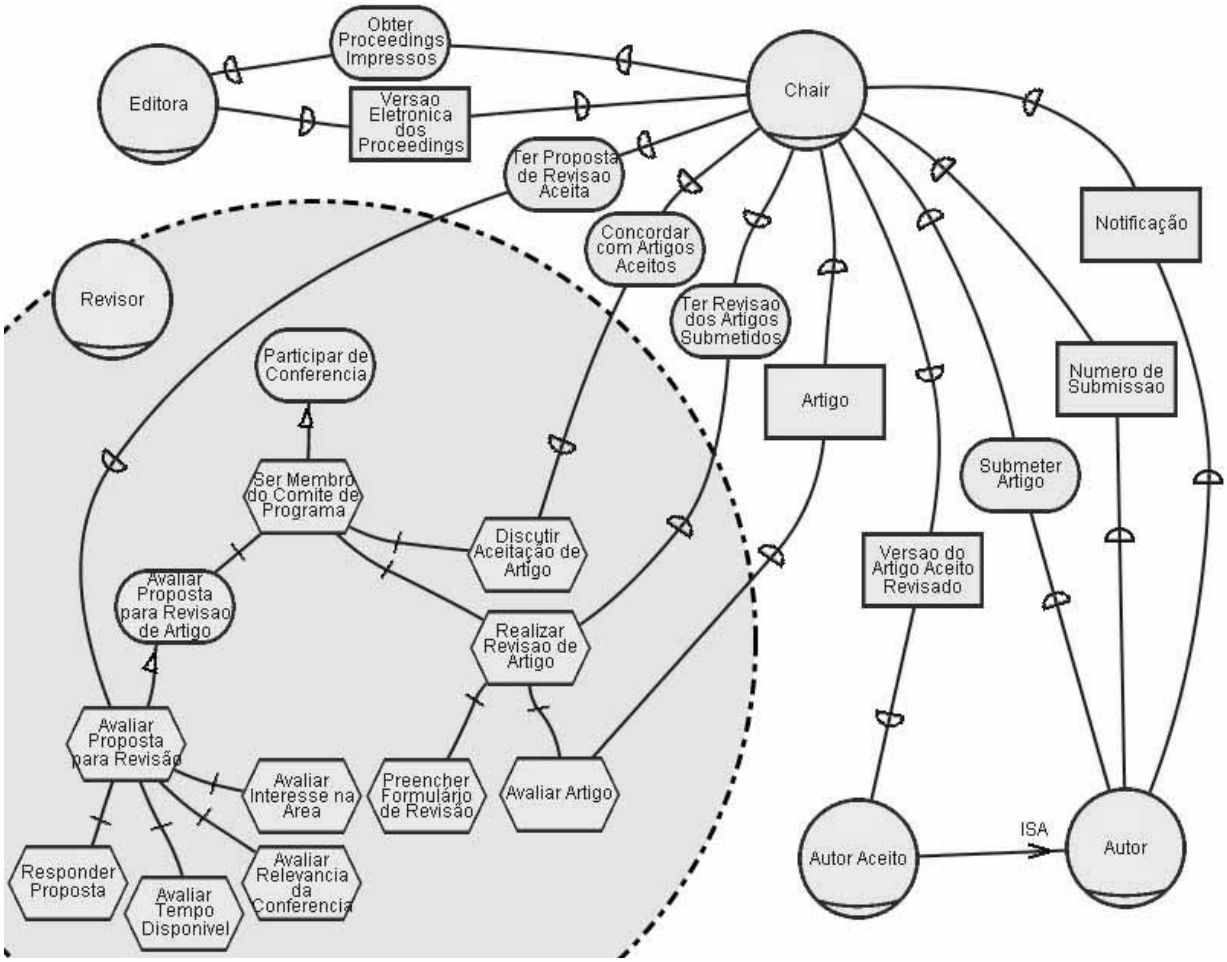


Figura 12 - Exemplo gráfico do SR (revisor) na fase de Requisitos Iniciais.

A *Figura 13* ilustra a análise das intenções do ator *Chair* que tem como objetivo principal *Realizar Conferência*. Para atingir esse objetivo o *Chair* precisa *Coordenar uma Conferência*, tal tarefa compreende *Ter Artigos Submetidos*, *Ter Revisões Submetidas*, realizar *Notificação de Artigos Aceitos*, e *Publicar Proceedings* da conferência. Cada um desses objetivos é realizado através do

gerenciamento das fases de submissão (tarefa *Gerenciar Fase de Submissão*), revisão (tarefa *Gerenciar Fase de Revisão*), notificação (tarefa *Gerenciar Fase de Notificação*) e publicação (tarefa *Gerenciar Fase de Publicação*), respectivamente.

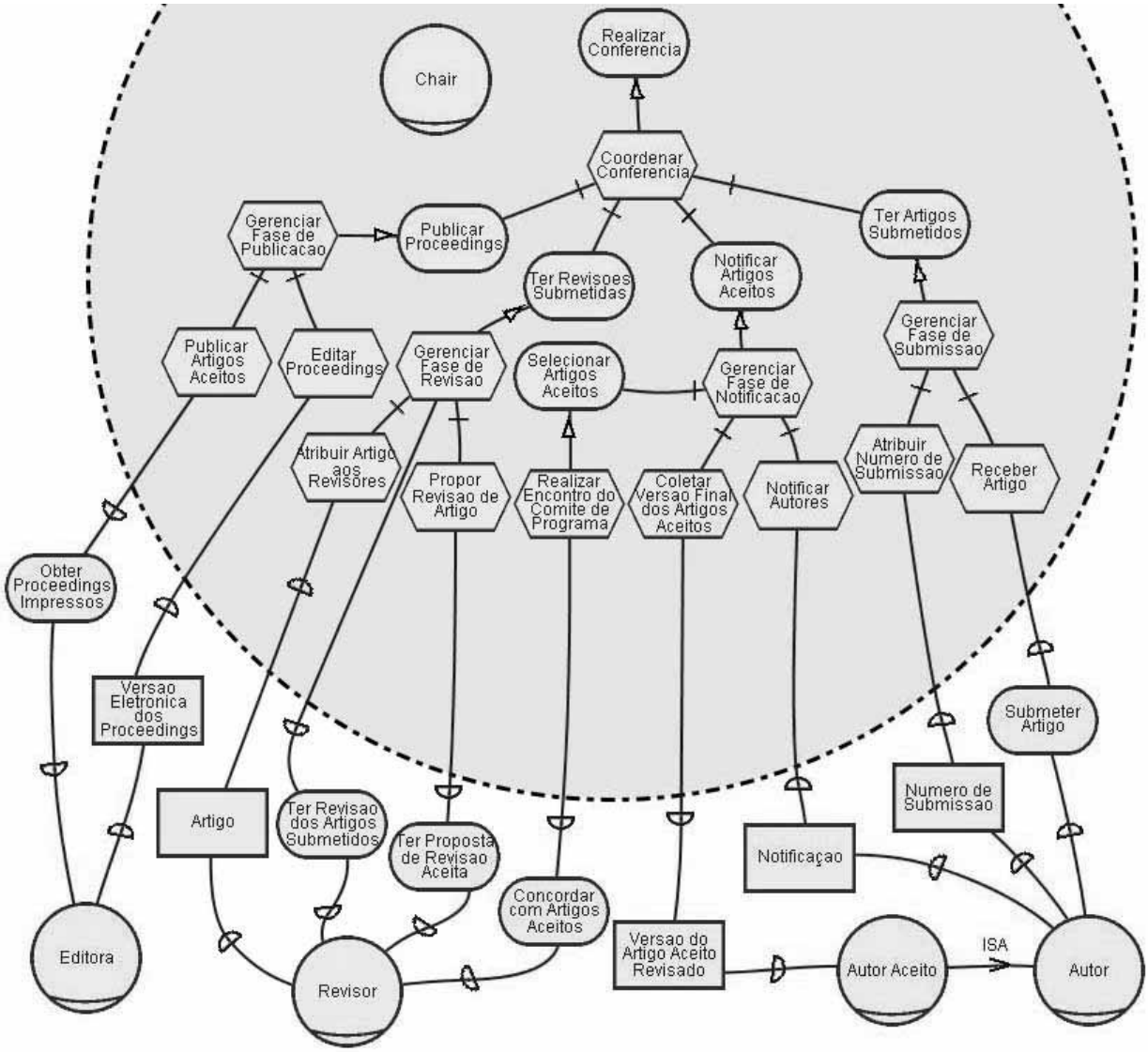


Figura 13 - Exemplo gráfico do SR (chair) na fase de Requisitos Iniciais.

Cada uma dessas fases compreende um conjunto de tarefas a serem realizadas (vide *Figura 13*):

(i) Fase de Submissão: fase onde deve-se *Receber Artigos* e *Atribuir Número de Submissão* aos artigos submetidos. O ator Autor depende dessas duas tarefas para que seu objetivo de *Submeter Artigo* seja atingido e que o recurso *Número de Submissão* seja disponibilizado, respectivamente.

(ii) Fase de Revisão: fase onde deve ser realizada a proposta de revisão dos artigos para os revisores (tarefa *Propor Revisão de Artigo*) e a distribuição dos artigos para os revisores (tarefa *Atribuir Artigo aos Revisores*). As tarefas da fase de revisão contribuem para que os objetivos do ator *Chair - Ter Revisão dos Artigos Submetidos* e *Ter Proposta de Revisão Aceita* - sejam atingidos. E o ator *Revisor* depende das atividades da fase de revisão para que o recurso *Artigo* seja disponibilizado.

(iii) Fase de Notificação: fase tem o objetivo de *Selecionar os Artigos Aceitos* - atingido pela atividade de *Realizar Encontro do Comite de Programa,* e compreende as atividades de *Notificar Autores* sobre a aceitação ou não dos artigos e *Coletar a Versão Final dos Artigos Aceitos*. Nessa fase o Chair disponibiliza ao autor que teve seu artigo aceito pela conferência *Notificação*, e depende do autor para que a *Versão do Artigo Aceito Revisado* seja disponibilizada.

(iv) Fase de Publicação: esta fase compreende as tarefas *Editar Proceedings* e *Publicar Artigos Aceitos*. A *Editora* depende desta primeira tarefa para que o recurso *Versão Eletrônica dos Proceendings* seja disponibilizado. E a última tarefa faz parte do objetivo do *Chair* de *Obter Proceedings Impressos*.

Requisitos Finais

A fase de Requisitos Finais se preocupa com o refinamento dos modelos (SD e SR) produzidos na fase de requisitos iniciais. Neste refinamento os modelos são revisados e é introduzido o sistema a ser desenvolvido como outro ator no modelo de dependência estratégica bem como sistemas e atores externos à organização.

O ator que representa o sistema é relacionado aos atores sociais em termos de dependências. Este deverá ser expandido para serem identificadas as razões por trás de suas dependências. Suas razões são analisadas e detalhadas (através de ligações meio-fins e de decomposição de tarefa) e irão eventualmente levar a revisar e adicionar novas dependências com um subconjunto de atores sociais (os usuários). As razões (objetivos) do ator sistema serão operacionalizadas através de tarefas dos demais atores.

Se necessário decompõe-se o ator que representa o sistema em vários atores e deve-se então revisar os modelos de razão e dependência estratégica desde a fase de requisitos iniciais.

As figuras 14 e 15 exemplificam o uso de modelos em i*, ilustrando respectivamente o modelo SD e o modelo SR nesta fase de requisitos final do Tropos.

Em nosso exemplo o sistema *Conference Management System* foi introduzido no modelo SD, como pode ser visualizado na *Figura 14*, não foi identificada a necessidade de introdução de outros atores ou sistemas externos. Podem ser visualizadas na figura as dependências entre os atores *Autor*, *Chair* e *Revisor* com o sistema. O *Autor* depende do *Conference Management System* para atingir seu objetivo de *Submeter Artigo* e desta forma ter à sua disponibilidade um *Número de Submissão*. Em relação ao sistema tanto o *Autor* quanto os revisores e o *Chair* esperam que os dados possam ser armazenados de forma a garantir a *Segurança* e *Integridade*, e que o sistema esteja sempre disponível. O *Chair* também depende do sistema para atingir os objetivos de gerenciar as fases de submissão e revisão, e de ter a *Revisão dos Artigos* disponível. O *Revisor* depende do sistema para *Avaliar Proposta de Revisão* e de ter o *Artigo* disponível para realizar a revisão. O sitema ainda depende do *Revisor* para *Configurar o Perfil do Revisor* montando assim um cadastro do revisor contendo área de pesquisa, dentre outros dados, e *Ter Revisão dos Artigos* para que o sistema possa disponibilizar as revisões para o *Chair*.

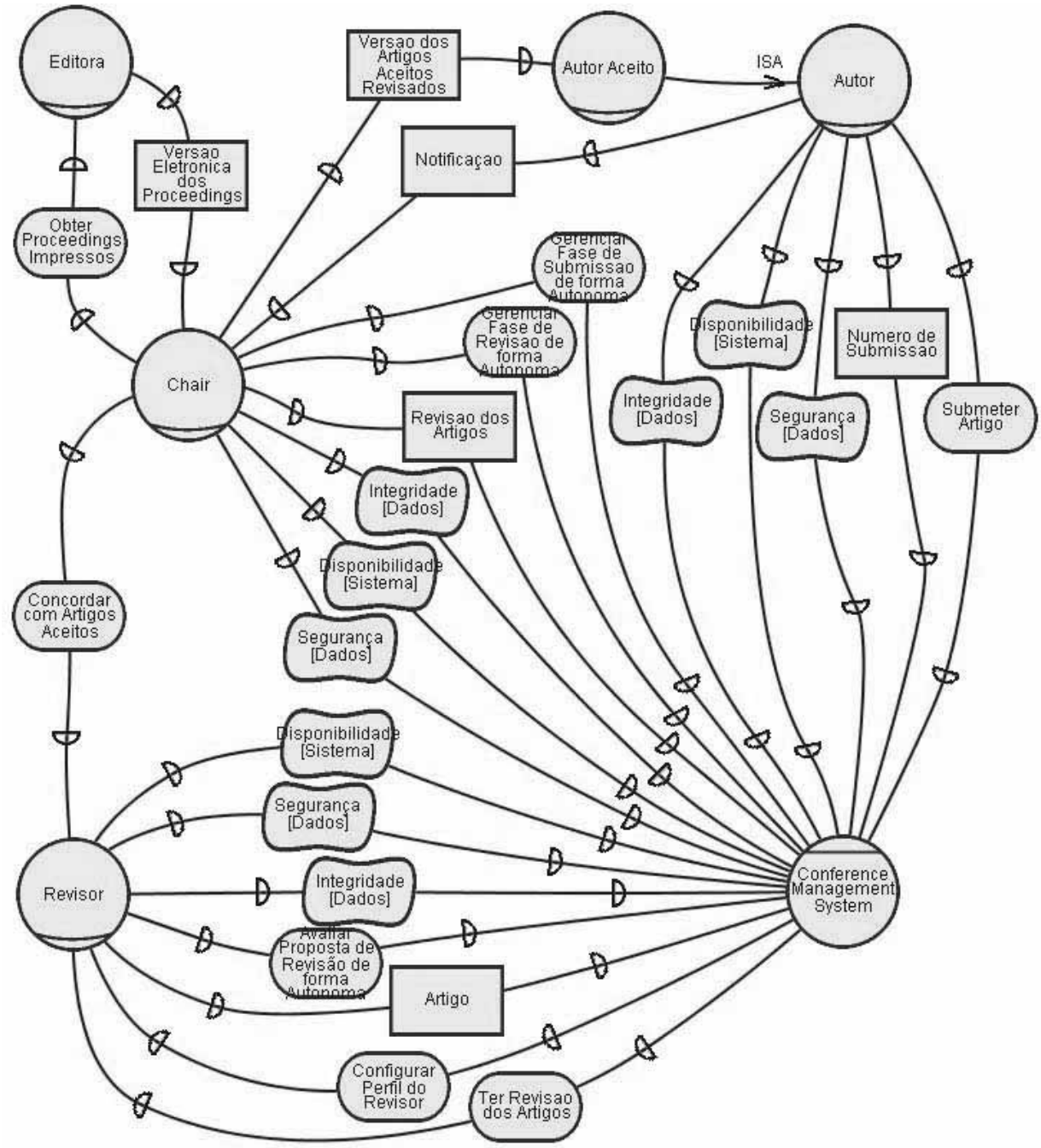


Figura 14 - Exemplo gráfico do SD na fase de Requisitos Finais.

No modelo SR, *Figura 15*, temos a análise das intenções do *Conference Management System* - uma análise detalhada das razões existentes, ou intencionalidades, por trás das dependências entre os atores *Chair*, *Autor*, *Autor Aceito*, *Revisor* e *Editora*, com o sistema. O objetivo principal do *Conference Management System* é *Automatizar os Serviços da Conferência* a fim de que as fases que envolvem o evento sejam automatizadas. Para que este objetivo seja atingido o sistema deve envolver as atividades de *Gerenciar Fase de Submissão*, *Avaliar Proposa para Revisão* e *Gerenciar Fase de Revisão*.

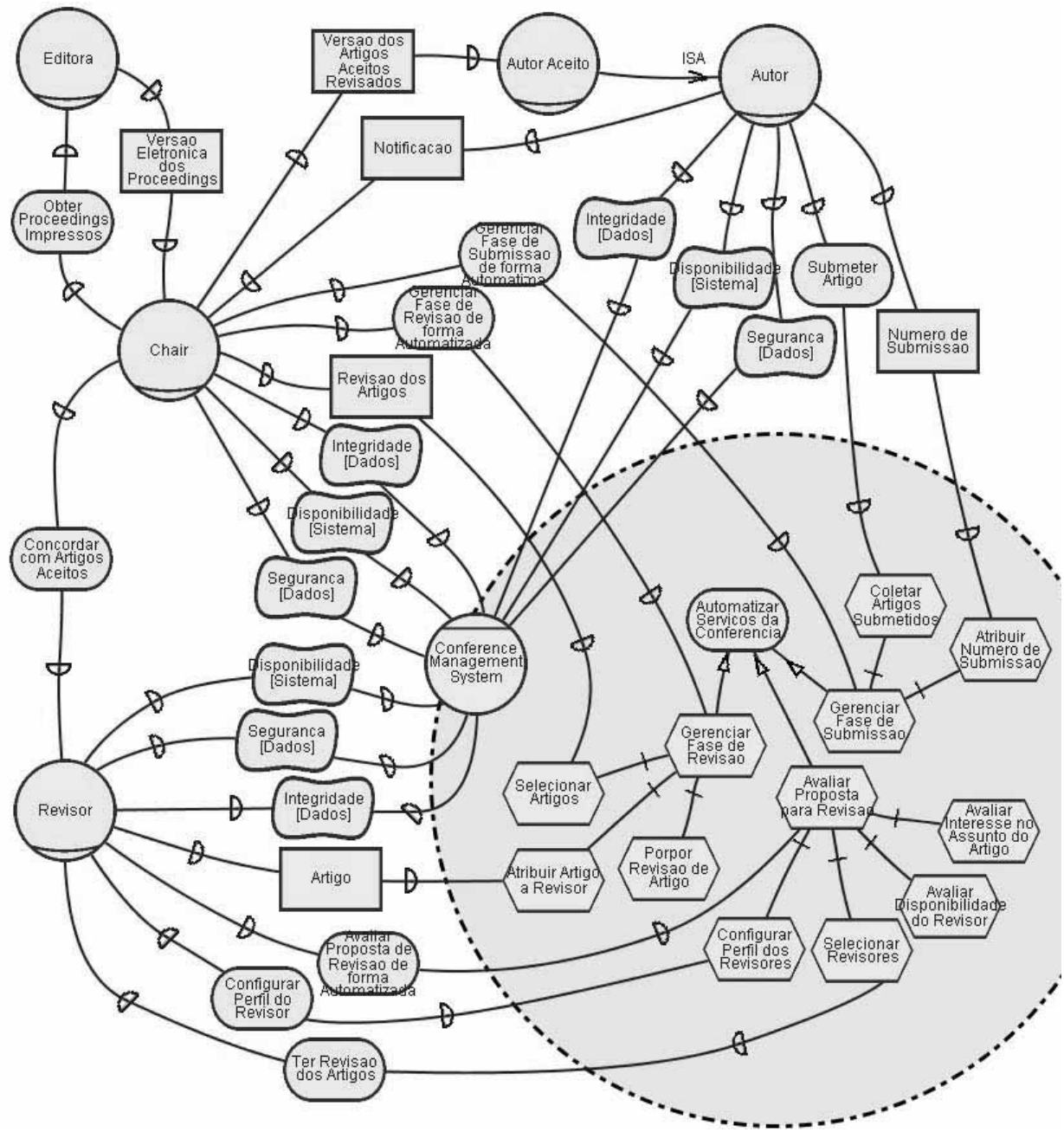


Figura 15 - Exemplo gráfico do SR na fase de Requisitos Finais.

Cada uma dessas atividades compreende um conjunto de tarefas a serem realizadas (vide *Figura 15*):

(i) Fase de Submissão: a atividade de *Gerenciar Fase de Submissão* atende ao objetivo do ator *Chair* de *Gerenciar Fase de Submissão de forma Automatizada*. Esta é a fase onde a coleta dos artigos submetidos é realizada, e os números de submissão são atribuídos. O ator Autor depende dessas duas tarefas - *Coletar Artigos Submetidos* e *Atribuir Número de Submissão* - para que seu objetivo de *Submeter Artigo* seja atingido e que o recurso *Número de Submissão* seja disponibilizado, respectivamente.

(ii) Avaliação de Proposta para Revisão: a atividade de *Avaliar Proposta para Revisão* atende ao objetivo do ator *Revisor* de *Avaliar Proposta de Revisão de forma Automatizada*. Esta atividade compreende as atividades de *Configurar Perfil dos Revisores*, *Avaliar Interesse no Assunto do Artigo*, *Avaliar Disponibilidade do Revisor* e *Selecionar Revisores*.

(iii) Fase de Revisão: a atividade de *Gerenciar Fase de Revisão* atende ao objetivo do ator *Chair* de *Gerenciar Fase de Revisão de forma Automatizada*. Esta é a fase onde ocorrem as atividades de *Selecionar Artigos*, *Atribuir Artigo a Revisor* e *Propor Revisão de Artigo*.

Projeto Arquitetural

A arquitetura de software de um sistema consiste dos componentes de software (subsistemas), suas propriedades externas (interconexões através de fluxos de controle de dados), e seus relacionamentos com outros softwares. Em Tropos, subsistemas são representados como atores, e as interconexões de dado/controle são representadas como dependências entre os atores [Castro 2006].

Um estilo arquitetural define uma família de sistemas relacionados em termos de um padrão de organização estrutural [Shaw e Garland 1993]. Em [Kolp 2006] são apresentados estilos arquiteturais organizacionais para auxiliar o projeto da arquitetura de aplicações cooperativas, dinâmicas e distribuídas, tais como sistemas multi-agente, no contexto do projeto Tropos. Os estilos arquiteturais organizacionais (*pyramid*, *joint venture*, *structure in* 5, *takeover*, *arm's length*, *vertical integration*, *co-optation*, *bidding*, ...) são estruturas genéricas que podem ser instanciadas para o projeto da arquitetura de uma aplicação específica. Atualmente, Tropos utiliza a notação do *framework* i* [Yu 1995] para representar os modelos arquiteturais definidos por [Kolp 2006], mas segundo [Silva, C. 2003] é inadequado utilizar i* como uma linguagem de descrição arquitetural (ADL - *Architecture Description Language*), pois o i* apresenta algumas limitações sobretudo no que diz respeito ao detalhamento comportamental do projeto arquitetural.

Esta fase inicia com a escolha do estilo arquitetural. Isso ocorre utilizando como critério de seleção as qualidades (objetivos-*soft*) desejadas que foram identificadas na fase anterior. Esta análise deverá ser guiada pelo Catálogo de Correlação [Kolp 2001], apresentado no quadro (*Tabela 4*) abaixo, que resume a avaliação dos estilos organizacionais na satisfação dos atributos de qualidade. A notação de contribuição faz referência ao *framework* NFR (*Non-Functional Requirements*) [Chung 2000] e tem a legenda apresentada na *Tabela 5*.

O estilo arquitetural é selecionado e aplicado ao projeto arquitetural do sistema. Uma vez escolhido o estilo arquitetural, a inclusão de novos atores e dependências, assim como a decomposição dos atores e das dependências existentes em outros atores e outras dependências poderá acontecer. Neste caso, deverá ser realizada a revisão dos modelos de razão e dependência estratégica.

Tabela 4 - Catálogo de Correlação, [Kolp 2001].

Atributos de Qualidade									
Estilos	Predicabilidade	Segurança	Adaptabilidade	Cooperatividade	Competitividade	Disponibilidade	Integridade	Modularidade	Agregabilidade
Flat Structure	--	--	-			+	+	++	-
Structure in 5	+	+		+	-	+	++	++	++
Pyramid	++	++	+	++	-	+	--	-	
Joint-Venture	+	+	++	+	-	++		+	++
Bidding	--	--	++	-	++	-	--	++	
Takeover	++	++	-	++	--	+		+	+
Arm's-Length	-	--	+	-	++	--	++	+	
Hierarchical Contracting			+	+	+	+		+	+
Vertical Integration	+	+	-	+	-	+	--	--	--
Co-optation	-	-	++	++	+	--	-	--	

Tabela 5 - Legenda de contribuições para a avaliação dos atributos de qualidade.

Notação	***Contribuições***
+	Parcial/Positivo
++	Suficiente/Positivo
-	Parcial/Negativo
--	Suficiente/Positivo

Projeto Detalhado

A fase Projeto Detalhado visa especificar o nível micro de agente, detalhando capacidades e planos, protocolos de comunicação e coordenação. É proposto em [Silva, C. T. L. L. 2007] um processo a ser seguido na fase de projeto detalhado que envolve dois papéis (Projetista de Agentes e o Arquiteto de Software), três guias (Meta-modelo de agência, *framework* i*, e eurísticas de mapeamento), e dez artefatos de saída (seis modelos UML, dois modelos SMA, e três documentos), ilustrados na *Figura 16*.

De acordo com a *Figura 16*, cada papel do processo executa atividades específicas que podem utilizar guias específicos para consumer e/ou produzir diferentes artefatos. O Projetista de Agentes é responsável por escolher o modelo de agente mais apropriado para a arquitetura interna [Ferber e Gutknecht 2000]. O Arquiteto de Software é uma pessoa, time, ou organização responsável pela arquitetura do sistema [IEEE 2000].

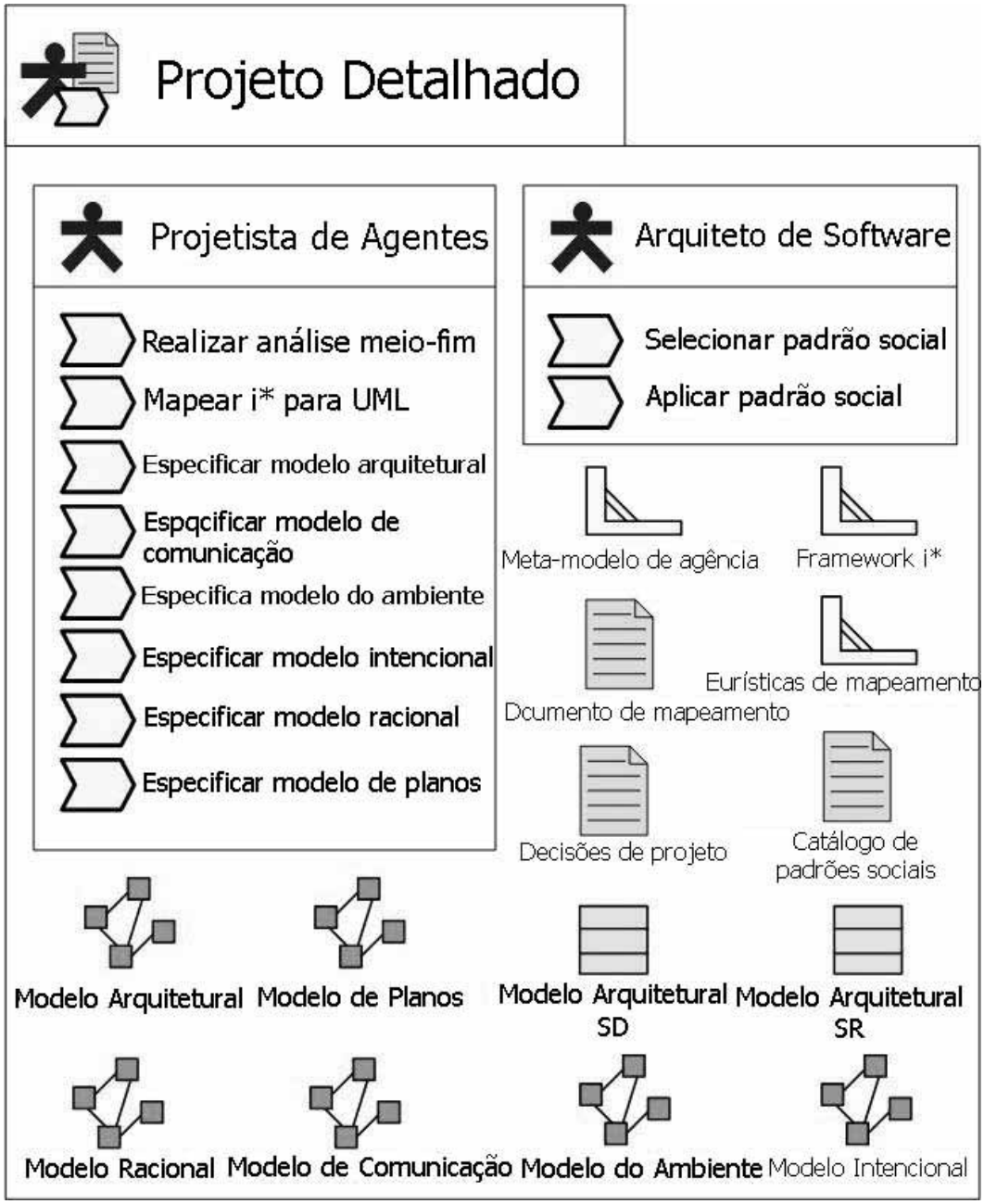


Figura 16 - Projeto Detalhado proposto por [Silva, C. T. L. L. 2007] .

Após a fase de projeto detalhado, algumas diretrizes podem ser utilizadas para a implementação do sistema que foi modelado. Implementar sistemas multi-agente é inerentemente difícil [Castro 2006]. Por esta razão, muitas plataformas de implementação têm sido construídas para simplificar o desenvolvimento de sistemas multi-agente. Em geral, estas plataformas são utilizadas tanto para desenvolvimento quanto como um *middleware* onde os agentes podem rodar.

Como exemplo de ambientes disponíveis para suportar a implementação de sistemas multi-agente, apresentaremos três plataformas que estão em conformidade com a FIPA (*Foundation for Intelligente Physical Agents*) [FIPA 2007] e totalmente implementadas em Java [JAVA 2008]:

(i) FIPA-OS [FIPA-OS 2007] é uma plataforma baseada em componentes que possibilita o rápido desenvolvimento de agentes. FIPA-OS suporta a maioria das especificações experimentais da FIPA e está sendo continuamente melhorada como um projeto *open source* gerenciado.

(ii) JACK Intelligent Agents [JACK 2007] é um ambiente de desenvolvimento orientado a agentes projetado para estender Java com o modelo de agente BDI [Rao e Georgeff 1995]. Para suportar a programação de agentes BDI (*Beliefs, Desires and Intentions*), JACK oferece cinco principais elementos da linguagem: agentes, capacidades, relações de banco de dados, eventos e planos. Capacidades agregam eventos, planos, bancos de dados ou outras capacidades, cada uma delas assumindo uma função específica atribuída a um agente. Relações de banco de dados armazenam crenças e dados de um agente. Eventos identificam as circunstâncias e mensagens a que um agente pode responder. Planos são instruções.

(iii) JADE Framework (Java *Agent DEvelopment framework*) [JADE 2007] é um *middleware* que facilita o desenvolvimento de sistemas multi-agentes conforme os padrões da FIPA. Em JADE, um comportamento representa uma tarefa que um agente pode executar. Uma das mais importantes características dos agentes JADE é a habilidade de se comunicar. Mensagens trocadas por agentes JADE têm um formato especificado pela linguagem ACL (*Agent Communication Language*) definida pela FIPA [FIPA 2007]. Este formato compreende um número de campos e, em particular, o campo de performativa que indica a intenção comunicativa que o agente que envia a mensagem pretende atingir ao enviar a mensagem. O serviço de páginas amarelas em JADE (de acordo com a especificação da FIPA) é fornecido por um agente chamado DF (*Directory Facilitator*). JADE também fornece suporte para linguagem de conteúdo e ontologias que, entre outras coisas, verifica se as mensagens sendo trocadas estão de acordo com as regras da ontologia definida. Uma extensão desta ferramenta - JADEX [JADEX 2007], adiciona às características deste *framework* conceitos da arquitetura BDI (*Beliefs, Desires and Intentions*) de agentes.

3.2 Ferramentas de Suporte

Esta seção trata de ferramentas de apoio à modelagem segundo o i*. A maioria das ferramentas que serão aqui apresentadas dá suporte apenas à criação dos modelos segundo o *framework* i* enquanto que outras se propõe ao suporte das atividades inerentes as fases do Tropos. Nosso foco será identificar se estas estão de acordo com o i* segundo [i* Wiki 2008][Grau 2008] e se estas aplicam restrições na geração dos modelos.

3.2.1 GR-Tool

GR-Tool (GR – *Goal Reasoning*) [GR-Tool 2004] é uma ferramenta gráfica através da qual é possível: desenhar os modelos baseados em objetivos e realizar análise dos relacionamentos entre os objetivos (quantitativa e qualitativamente).

A análise qualitativa (*forward reasoning*) responde perguntas como: dado um modelo, e assumindo que os objetivos folhas foram atingidos, todos os objetivos raiz foram satisfeitos? Enquanto que a

análise quantitativa (*backward reasoning*) retorna respostas como: dado um modelo, qual o conjunto de objetivos que juntos satisfaz a todos os objetivos raiz.

Parte da ferramenta (algoritmos de análise quantitativa e qualitativa) foi desenvolvida em Java. GR-Tool não é *open source* e não é possível extender a ferramenta, e os modelos gerados por ela são apenas modelos com objetivos e eventos (semelhante ao conceito de *tarefa* usado no i*) não envolvendo outros elementos que o i* contempla (como por exemplo, recurso ou objetivo *soft*). Além de não ser possível modelar segundo todos os conceitos do i*, a ferramenta não suporta a metodologia Tropos. A versão do i* na qual a ferramenta se baseia é o i* do Tropos [Bresciani 2004][Sannicoló 2002][Tropos Project 2008].

A *Figura 17* ilustra a ferramenta GR-Tool, podemos observar que na parte do menu superior da ferramenta encontram-se as opções de criação dos nós (objetivos e eventos), e no menu esquerdo observa-se as opções de análise aplicáveis aos modelos criados e a árvore de nós criados.

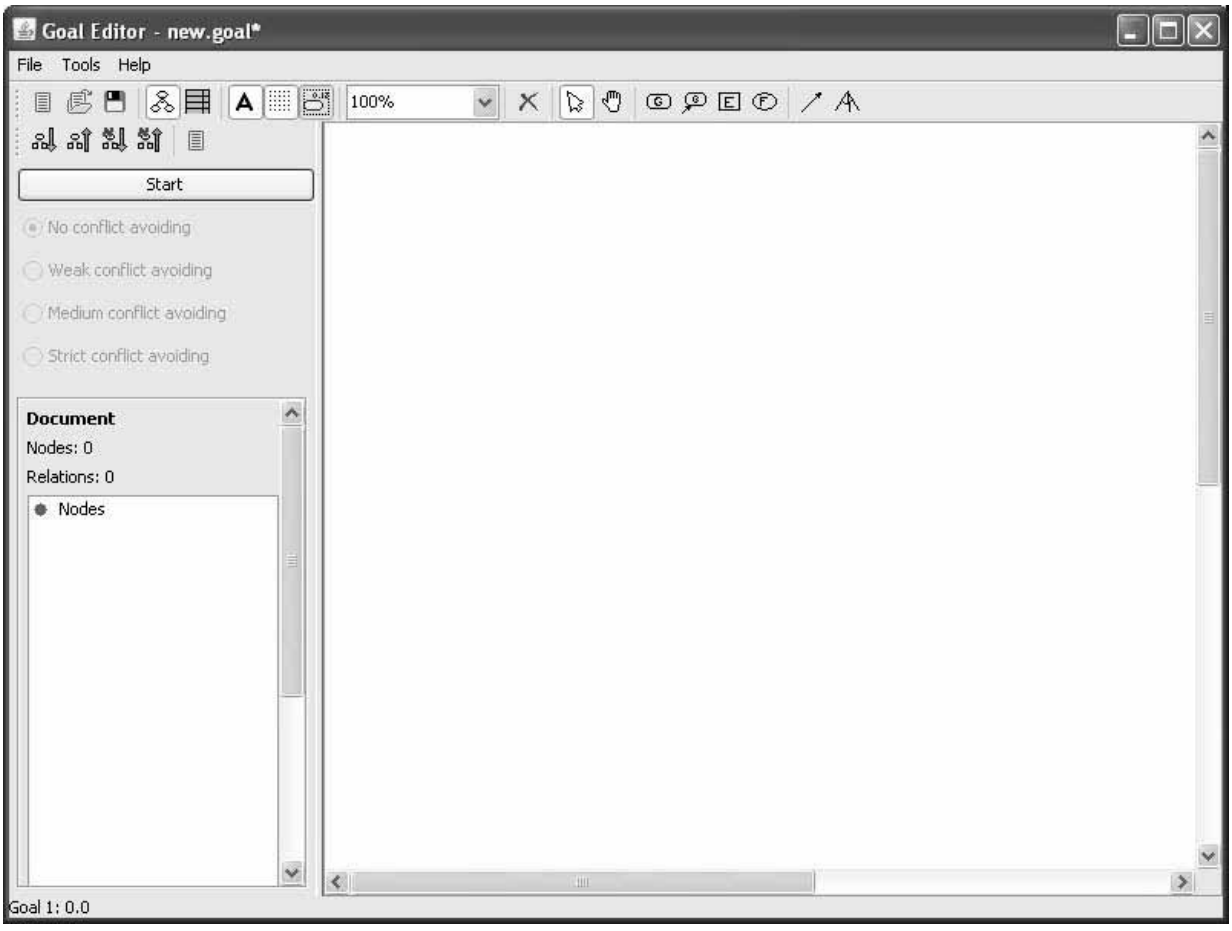


Figura 17 - Screenshot GR-Tool.

3.2.2 J-PRiM

J-PRiM (*a Java tool for a Process Reengineering i* Methodology*) [J-PRiM 2006][Grau 2006] é uma ferramenta que permite analisar um sistema de informação existente e representá-lo como uma hierarquia de elementos segundo o i* [Yu 1995]. Uma vez modelado, várias alternativas para o sistema

podem ser exploradas, cada uma delas modeladas como um modelo i* diferente. Todas as alternativas geradas podem ser avaliadas através da definição e aplicação de métricas sobre os modelos i* a fim de estabelecer qual a alternativa mais apropriada para representar o sistema. Na *Figura 18* abaixo, observamos um *screenshot* da ferramenta.

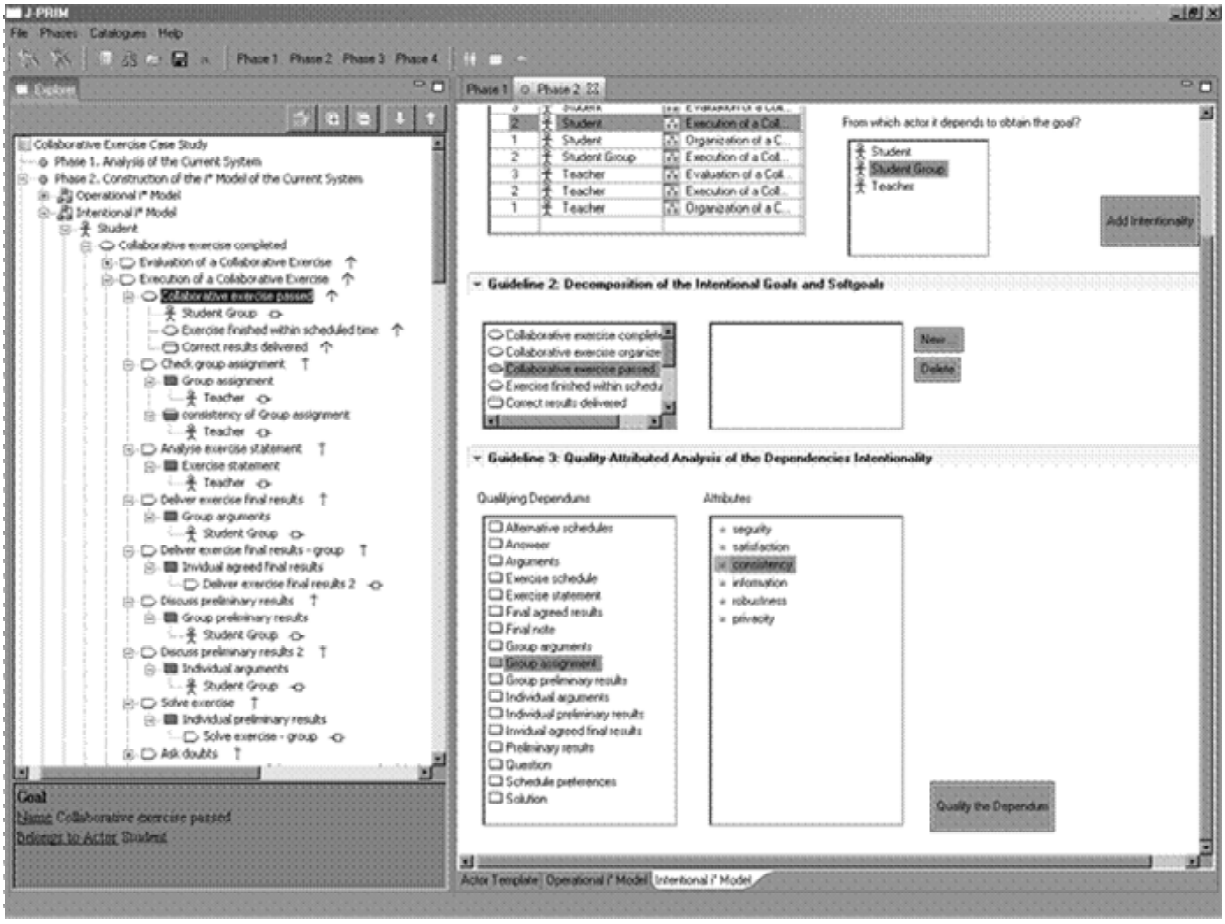


Figura 18 - Screenshot da ferramenta J-PRiM.

A ferramenta foi desenvolvida em Java e seu código é aberto permitindo sua extensão. J-PRiM não dá suporte a metodologia Tropos e sim a outra metodologia (PRiM) que não é orientada a agentes, e é baseado no i* original [Yu 1995]. Além disso, J-PRiM não provê a verificação dos modelos gerados. A *Figura 18* apresenta a ferramenta J-PRiM.

A metodologia PRiM aborda a modelagem e análise com i* [Yu 1995] da perspectiva de reengenharia do processo, onde a especificação de um novo sistema começa da observação do sistema atual e termina com a elaboração da especificação do novo sistema. PRiM é composta de 5 fases: (1) Análise do Processo Atual, (2) Construção do Modelo i*, (3) Geração de Alternativas, (4) Avaliação de Alternativas, (5) Especificação do novo sistema.

3.2.3 OME

OME (*Organizational Modeling Environment*) [Yu e Liu 2005] é uma ferramenta cujo propósito geral é ser um editor gráfico para suporte à modelagem orientada a objetivo e/ou orientada a agentes.

Esse ambiente de modelagem organizacional ajuda o usuário no desenvolvimento e representação de modelos desenvolvidos com os *frameworks* i* (i-*star*) e NFR (*Non-Functional Requirements*). A ferramenta OME está sendo desenvolvida no *Knowledge Management Laboratory*, na Universidade de Toronto e encontra-se na versão 3.

A tela inicial do OME (*Figura 19*) é um ambiente através do qual o usuário pode organizar os vários modelos desenvolvidos dentro de um projeto. O usuário pode criar os projetos, que são armazenados no diretório *Projects* da ferramenta OME, e os modelos podem ser criados criados selecionando-se o projeto no qual o modelo estará armazenado e qual *framework* será usado para criação dos modelos (vide *Figura 20*). A ferramenta quando instalada vêm com um conjunto de projetos exemplos como ilustrado na *Figura 19* e *Figura 21*.

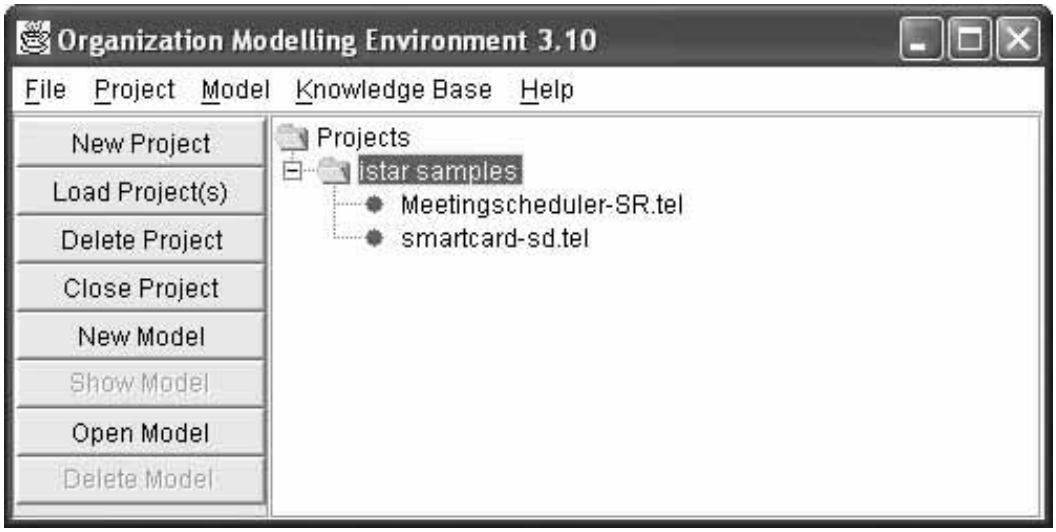


Figura 19 - Tela Inicial da ferramenta OME [Yu e Liu 2005].

OME é de fácil utilização, a maior parte dos recursos está de acordo com [Yu 1995]. A ferramenta dá suporte apenas à análise e modelagem seguindo o i* original de [Yu 1995] e não dando suporte às demais fases da metodologia Tropos, escolhida para nosso estudo. OME não permite a verificação dos modelos gerados e pode ser extendido através de *plug-in*.

Abaixo, na Figura 21, ilustramos a tela onde o usuário pode desenvolver os modelos segundo o framework i*. A figura apresenta um dos exemplos de modelos contidos na ferramenta e pode-se observar o menu com as opções para criação dos elementos i*: atores, elementos de dependência, links de dependência e links de associação. Para criar atores e elementos basta clicar na opção desejada e então clicar na área de trabalho, onde o modelo SD exemplo abaixo é visualizado. Para criar os links de dependência ou as associações deve-se clicar na opção desejada e então clicar primeiro, no elemento origem da relação, e por fim no elemento destino da relação. O OME valida as relações no momento em que estas estão sendo construídas permitindo links de dependência apenas entre atores e elementos de dependência, ou entre elementos de dependência, e permitindo associações apenas entre atores. Quanto aos links de decomposição de tarefa e ligação meio-fim o OME não realiza a validação e permite sua criação fora da fronteira de atores.

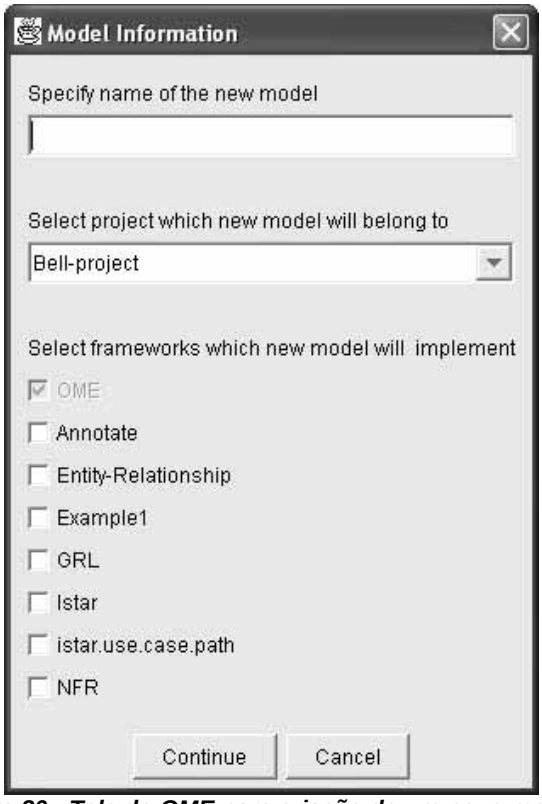


Figura 20 - Tela do OME para criação de um novo modelo.

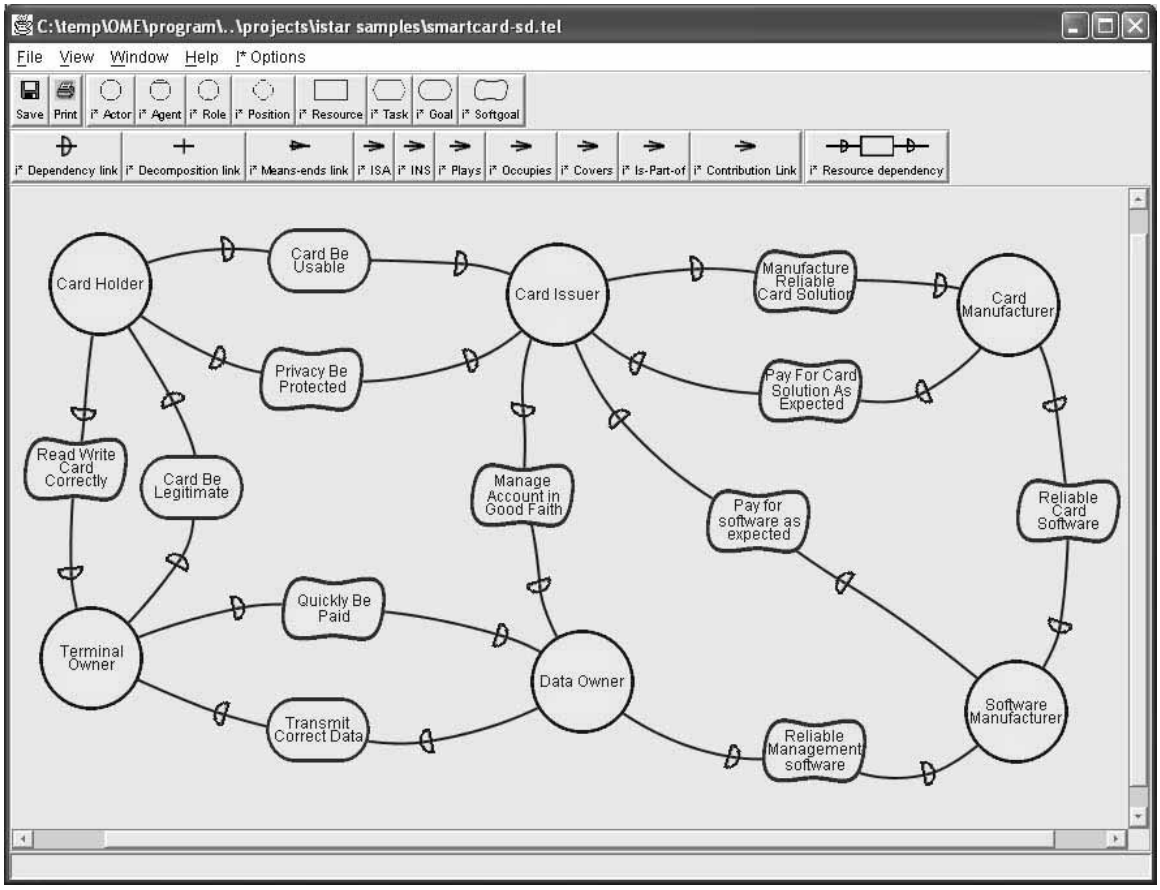


Figura 21 - Tela da ferramenta OME onde o usuário pode realizar modelagem segundo o i*.

3.2.4 OpenOME

A ferramenta OpenOME [OpenOME 2006] é um *plug-in* para o Eclipse [Eclipse 2008] e Protégé [Protégé 2007] que foi desenvolvido como uma ferramenta de propósito geral para modelagem e análise orientada a objetivos e/ou agentes. A ferramenta foi desenvolvida em Java e o termo *open* no nome da ferramenta se refere ao tipo do projeto - *open source*, através do qual, vários pesquisadores vem trabalhando para estender a ferramenta.

Segundo [OpenOME 2006] este *plug-in* está em conformidade com o i* original [Yu 1995], mas apesar de permitir a aplicação do *framework* i* o Tropos não é suportado por ele.

A *Figura 22* ilustra o *plug-in* sendo utilizado através do Eclipse. Assim como o OME (seção 3.2.3), o OpenOME vem com alguns exemplos de diagramas que podem ser acessados através do menu OpenOME, opção *Generate Example Project*. Na figura podemos observar 4 visões: uma lista com os projetos e diretórios que estão sendo trabalhados (*Project Explorer*), uma visão geral do modelo que está sendo construído (*Outline*), uma paleta com lista de opções para construção dos modelos (*Palette*), e a área de trabalho na qual o modelo é construído. O *plugin* OpenOME não fornece uma validação dos modelos gerados.

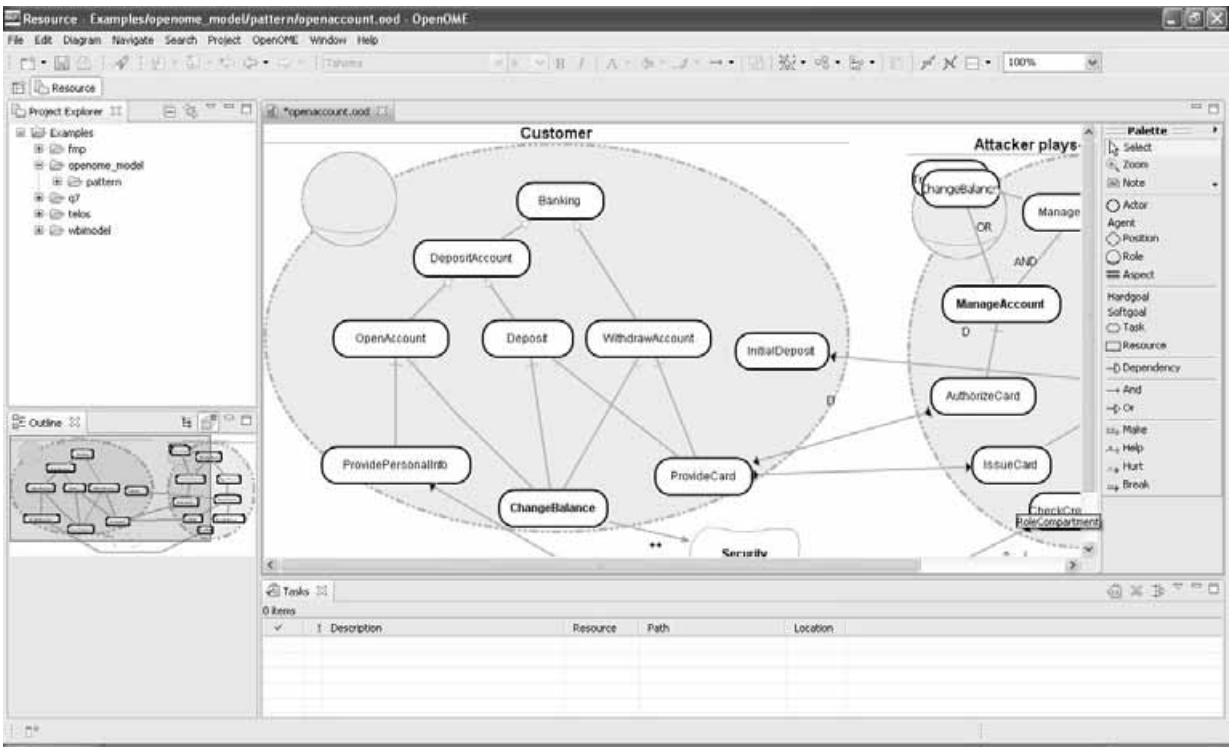


Figura 22 - Tela do Eclipse com o plug-in OpenOME.

3.2.5 REDEPEND-REACT

Redepend-React é uma ferramenta que provê suporte à modelagem i* segundo sua versão original [Yu 1995] e à análise dos modelos resultantes. A ferramenta é uma extensão da REDEPEND i* *modelling tool* [Grau 2005] que foca na representação do sistema de informação em análise usando o

framework i* e provendo funcionalidades para geração e avaliação de arquiteturas alternativas para o sistema modelado.

Segundo a *home page* do projeto [Redepend-React 2006] a ferramenta está de acordo com [Yu 1995]. Contudo, esta não suporta a metodologia Tropos, escolhida para nosso estudo.

As atividades suportadas pela ferramenta, ilustradas na *Figura 23*, envolvem:

(i) A construção do modelo i* - atividade pertinente ao Engenheiro de Requisitos,

(ii) A definição das propriedades e avaliação das arquiteturas propostas (baseadas no modelo i*) - atividade que envolve o Engenheiro de Requisitos e o Arquiteto de Software,

(iii) A construção de um catálogo das propriedades identificadas segundo o modelo i* - atividade pertinente ao Arquiteto de Software,

(iv) A construção de um catálogo de atores - atividade que envolve o Engenheiro de Requisitos, e

(v) A construção de um catálogo de componentes - que envolve o Analista.

Estes últimos itens de atividades (os catálogos gerados) são fornecidos a fim de promover reusabilidade, e retorno de investimento.

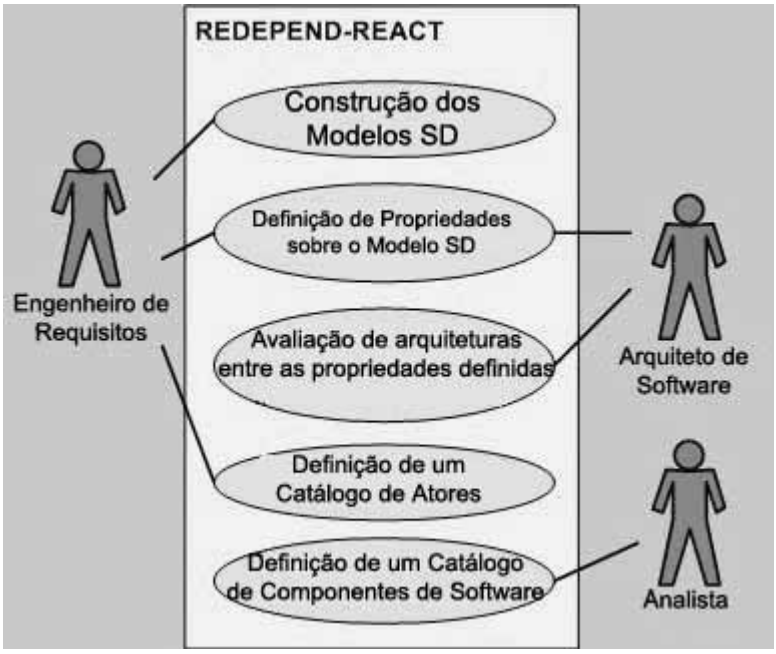


Figura 23 - Atividades do REDEPEND-REACT [Redepend-React 2006].

3.2.6 SNet Tool

A ferramenta SNet Tool é um trabalho da *RWTH Aachen University, Informatik 5* e do *Knowledge Based Systems Group* [i* Wiki 2008][Grau 2005]. Nesta ferramenta o i* é aplicado e estendido para prover suporte à redes inter-organizacionais.

A ferramenta fornece uma transformação automática da representação gráfica da rede (baseada no i* - uso do OME) em programas executáveis [i* Wiki 2008][Grau 2005]. Desta forma, participantes

da rede podem simular vários cenários cujos caminhos podem dar informações valiosas a cerca dos riscos e benefícios de tomar certas ações. A ferramenta destina-se a acompanhar a evolução da rede a fim de detectar problemas mais cedo e ajudar a rede a definir ações apropriadas para gerenciar esses problemas [Gans 2004].

Esta ferramenta apenas usa os conceitos do *framework* i* e não provê suporte ao Tropos. A *Figura 24* ilustra o simulador SNet Tool.

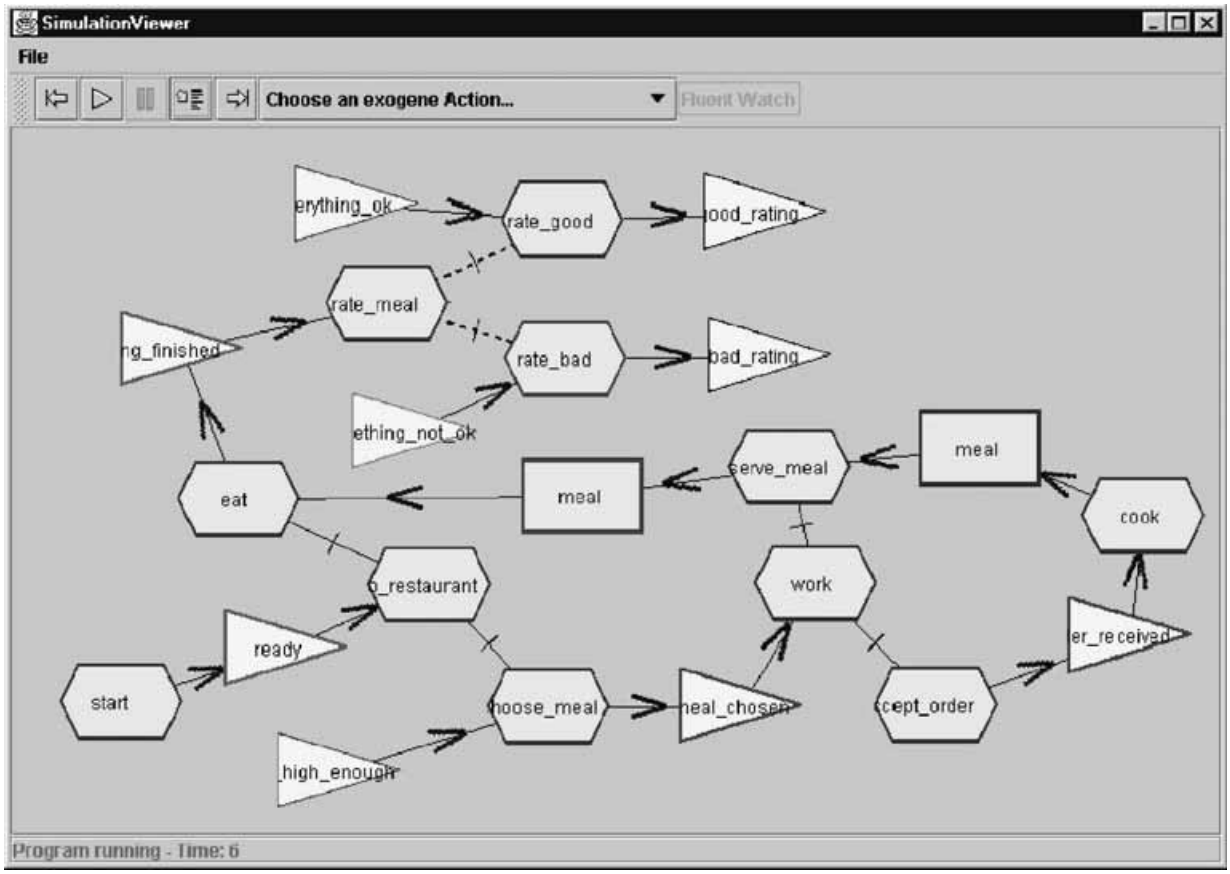


Figura 24 - Simulador SNet.

3.2.7 Si Tool*

O *Si* Tool* (*Security and Dependability Tropos Tool*) – também chamada de ST-Tool, é uma ferramenta gráfica onde é possível desenhar modelos em i*, e executar análises formais dos requisitos funcionais e de segurança [Giorgini 2005b]. A ferramenta foi desenvolvida em Java no *Department of Information and Communication Technology* da Universidade de Trento, na Itália [Si* Tool 2006]. Apesar de ser possível a geração de modelos i* a ferramenta não está de acordo com [Yu 1995] e não dá suporte a todas as fases do Tropos.

A *Figura 25* ilustra a ferramenta ST-Tool. Assim como na ferramenta OpenOME - apresentada na seção 3.2.4, a ST-Tool apresenta 4 visões através das quais se pode trabalhar:

(i) Uma lista com os projetos e diretórios que estão sendo trabalhados (Navigator),

(ii) Uma visão geral do modelo que está sendo construído (Outline),

(iii) Uma paleta com lista de opções - agrupadas em Ator, Serviços, Associaçãoes, Relacionamentos, Dependência, ECO Model - para construção dos modelos (Palette), e

(iv) Uma área de trabalho na qual o modelo é construído.

É interessante observar (*Figura 25*) que os modelos i* gerados nesta ferramenta não estão de acordo com uma das melhores práticas de representação dos elementos: os links de dependência não são representados com um 'D' apontando para seu destino de dependência como indicado em [i* Wiki 2008][Grau 2008] e sim como uma seta cheia.

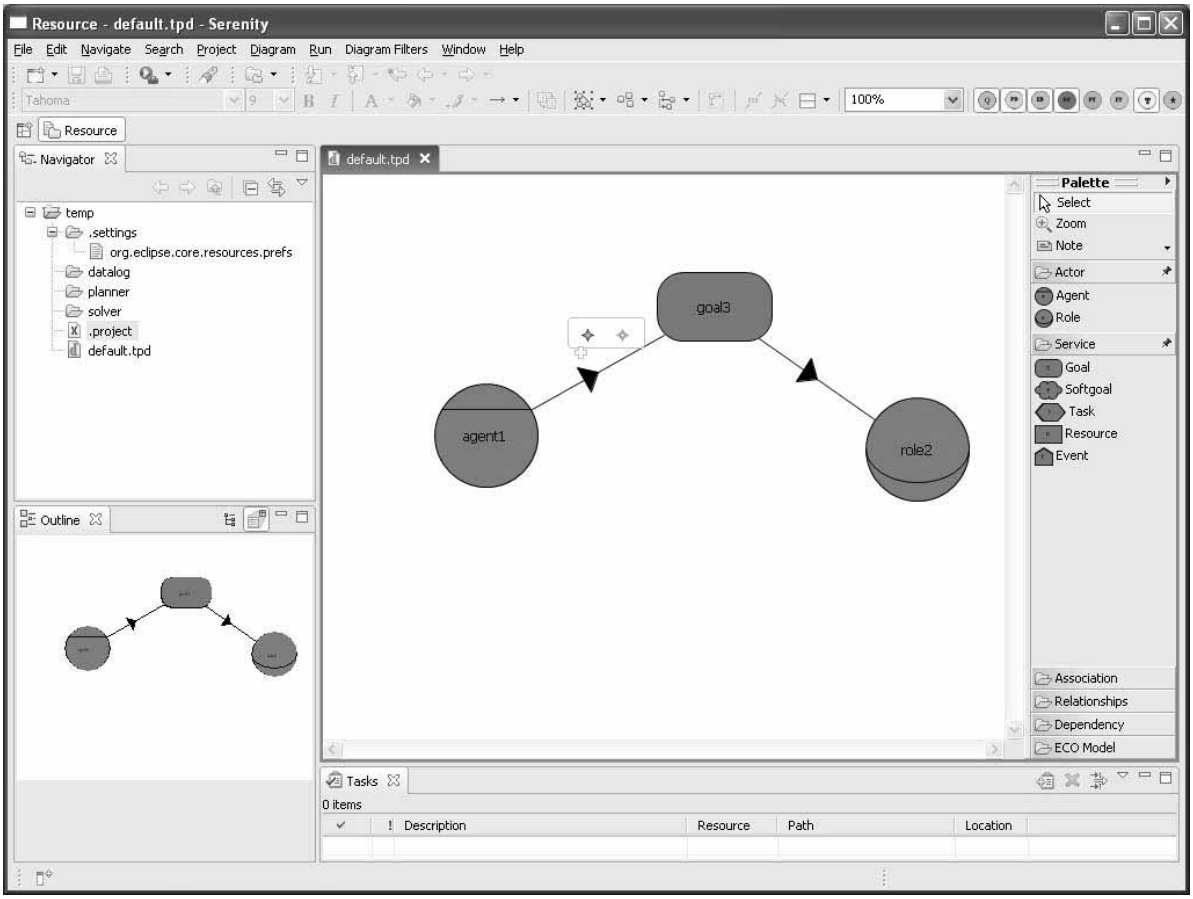


Figura 25 - Ferramenta Si* Tool.

3.2.8 T-Tool

T-Tool (*The Formal Tropos Tool*) fornece um *framework* para o uso de métodos formais na fase inicial de requisitos. A linguagem de especificação aceita é *Formal Tropos*, uma linguagem de especificação formal que oferece conceitos primitivos do i* (como atores, objetivos, e dependência entre atores).

Segundo [T-Tool 2003], a ferramenta permite diferentes tipos de análise. Por exemplo, ela permite a verificação se a especificação está consistente, ou se ela respeita um número de propriedades desejadas. Além disso, uma especificação pode ser ativada a fim de dar ao usuário um *feedback* imediato sobre suas implicações.

A *Figura 26* abaixo apresenta uma visão geral da análise feita utilizando a ferramenta T-Tool. Para suportar o mecanismo de análise das especificações de Formal Tropos, T-Tool aplica técnicas de verificação de modelo. Dessa forma, T-Tool é construída sobre uma ferramenta *open source* chamada NuSMV [Cimatti 2002] para checagem dos modelos. Como entrada a ferramenta recebe as especificações descritas em Formal Tropos (modelos gráficos e linguagem de especificação), as especificações são transformados para uma linguagem intermediária que serve de insumo para que o engenho de verificação (baseaso na NuSMV) realize uma análise formal. A saída da ferramenta é o resultado da verificação realizada sobre as especificações que indica se as possibilidades e afirmações dos requisitos formais satisfazem as restrições de criação e propriedades descritas nas especificações.

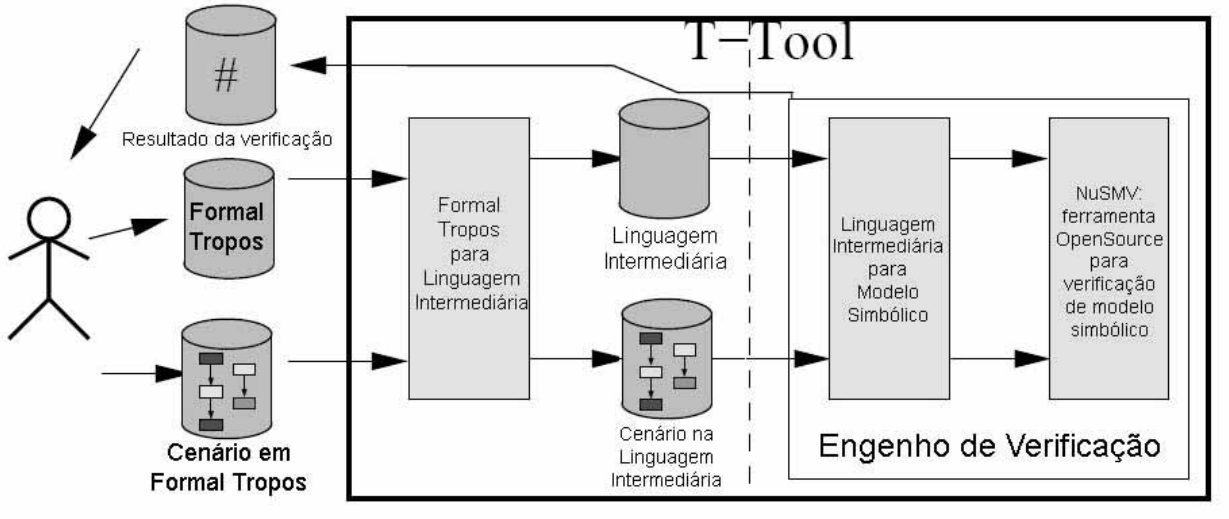


Figura 26 - Visão geral T-Tool.

T-Tool e *Formal Tropos* são parte de uma proposta de *framework* muito mais ampla, chamada Tropos, que como foi apresentada no início deste capítulo, propõe a aplicação de conceitos desde a fase de requisitos iniciais por todo o processo de desenvolvimento de software, incluindo as fases de requisitos finais, projeto arquitetural, projeto detalhado e implementação.

3.2.9 DesCARTES

DesCARTES (*Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems*) [DesCARTES Architect 2007] é uma ferramenta desenvolvida pela *ISYS Information Systems Research Unit*, da UCL (*Université Catholique du Louvain*). Seu propósito é suportar a edição de vários modelos: modelos i* (SD e SR), modelos NFR, modelos UML, modelos AUML no contexto do Tropos.

A ferramenta foi desenvolvida para ser um *plug-in* para o Eclipse e seu código-fonte não está acessível.

As figuras 27 e 28 mostram telas deste *plug-in*, a primeira exibe janelas com opções onde o usuário pode criar o modelo (diagrama) que desejar (i*, NFR, UML) e dizer a que fase do Tropos o modelo pertence. A *Figura 28* ilustra a interface onde o usuário usa de recursos gráficos para desenvolver os modelos. Assim como na ferramenta OpenOME - apresentada na seção 3.2.4, DesCARTES é baseada na plataforma Eclipse [Eclipse 2008] e possibilita a construção de modelos a partir da seleção da fase do Tropos à qual o modelo estará relacionado. Como pode ser visualizado na

Figura 28 a ferramenta apresenta uma visão com a lista de projetos e respectivos modelos (*Project Explorer*), uma visão onde os modelos são criados, e um menu superior com as opções de criação de elementos e relacionamentos de um modelo.

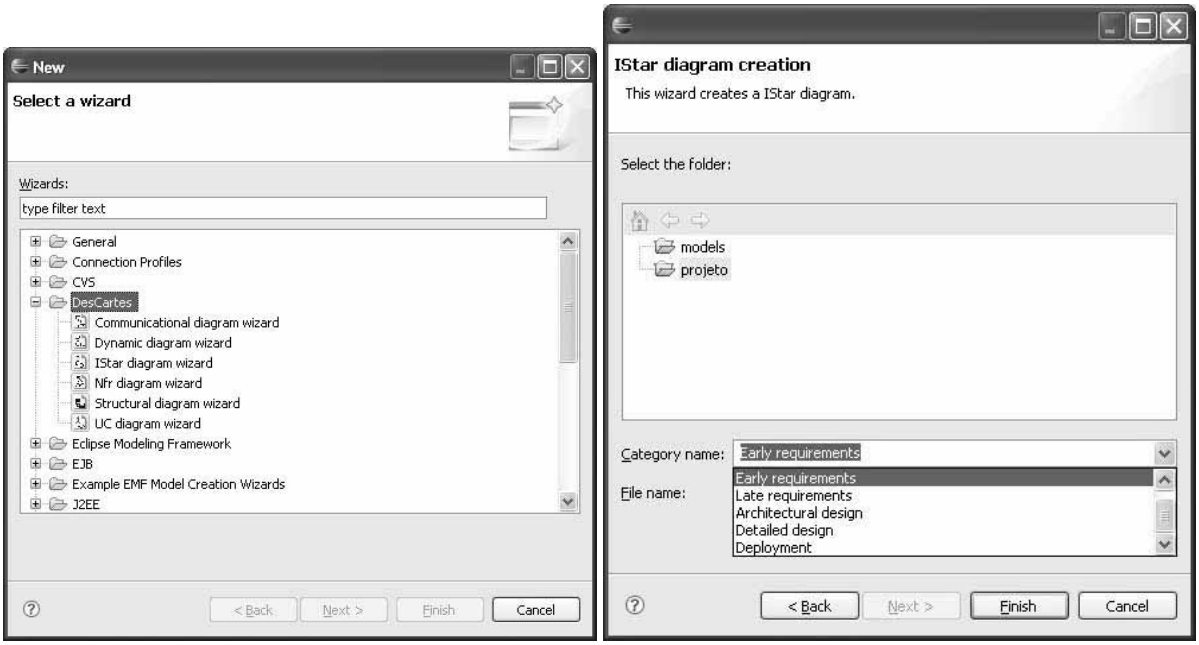


Figura 27 - Tela através da qual pode-se criar modelos no DesCARTES.

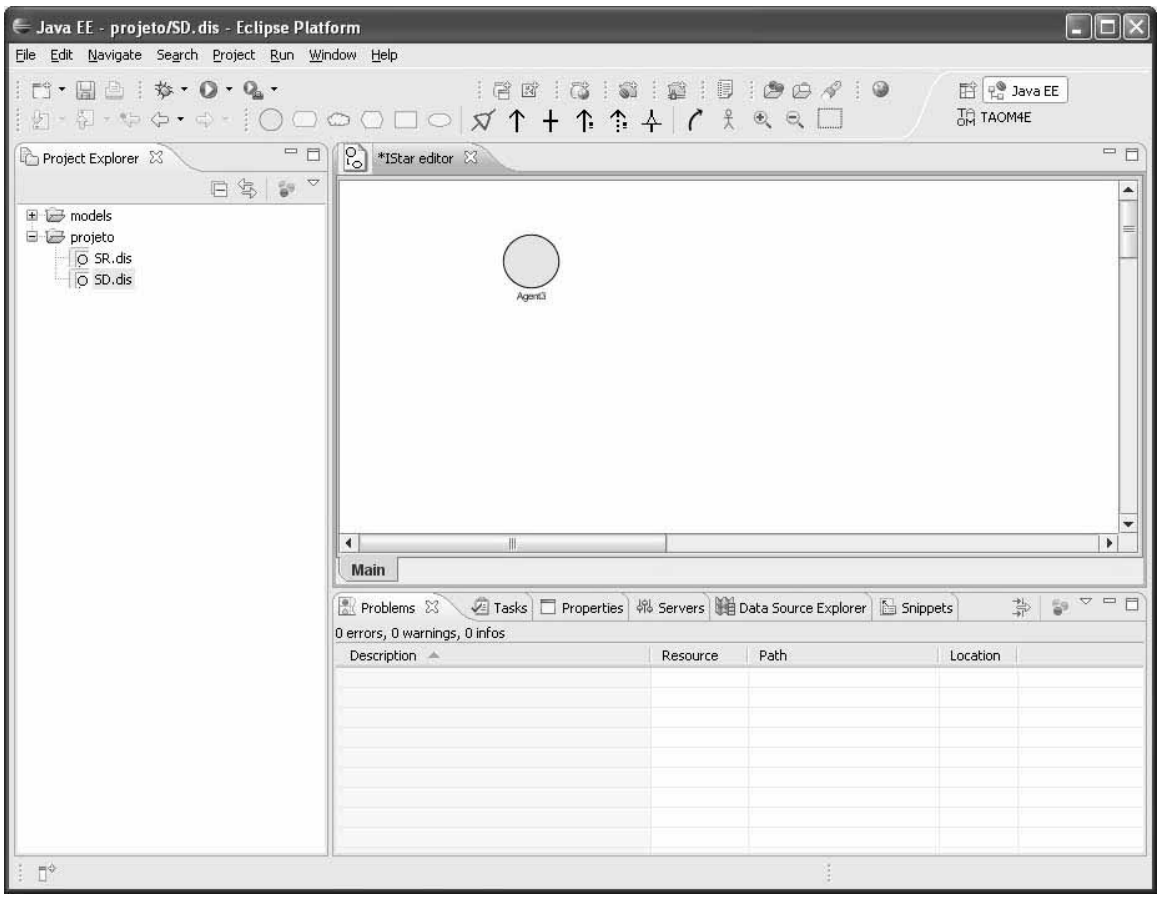


Figura 28 - Tela do DesCARTES onde se cria graficamente os modelos.

Segundo [DesCARTES Architect 2007] os modelos gerados estão em conformidade com o i* versão do Tropos [Bresciani 2004][Sannicoló 2002][Tropos Project 2008] e o i* original [Yu 1995].

A desvantagem do DesCARTES é que o código-fonte não está disponível. E, como veremos nos capítulos seguintes, existe uma evolução do i* [Yu 1995] sendo utilizada pela comunidade da Engenharia de Requisitos.

3.2.10 Visio Template

O Visio Template é uma extensão disponível em [i* Wiki 2008] cujo propósito é utilizar a ferramenta MS Visio - que é de propósito geral - aplicando esta extensão para representação dos diagramas/modelos em i*. É possível através do MS Visio criar extensões customizadas e utilizá-las para representação de diferentes tipos de diagramas/modelos como, por exemplo, o *framework* i*.

O problema neste caso é que, como não há verificação de conformidade do modelo, torna-se possível criar quaisquer formatos de diagramas. E para o nosso estudo a ferramenta também não provê o suporte ao Tropos.

A *Figura 29* ilustra o uso do *template* para o MS Visio. Adicionando-se a extensão disponível em [i* Wiki 2008] é possível através de uma lista de opções (como pode ser visto na figura abaixo ao lado esquerdo), criar os modelos i*. A extensão como pode ser visualizada permite a criação de atores, elementos de dependência, links de dependência, decomposição de tarefas, links meio-fim, links de associação, e links de contribuição.

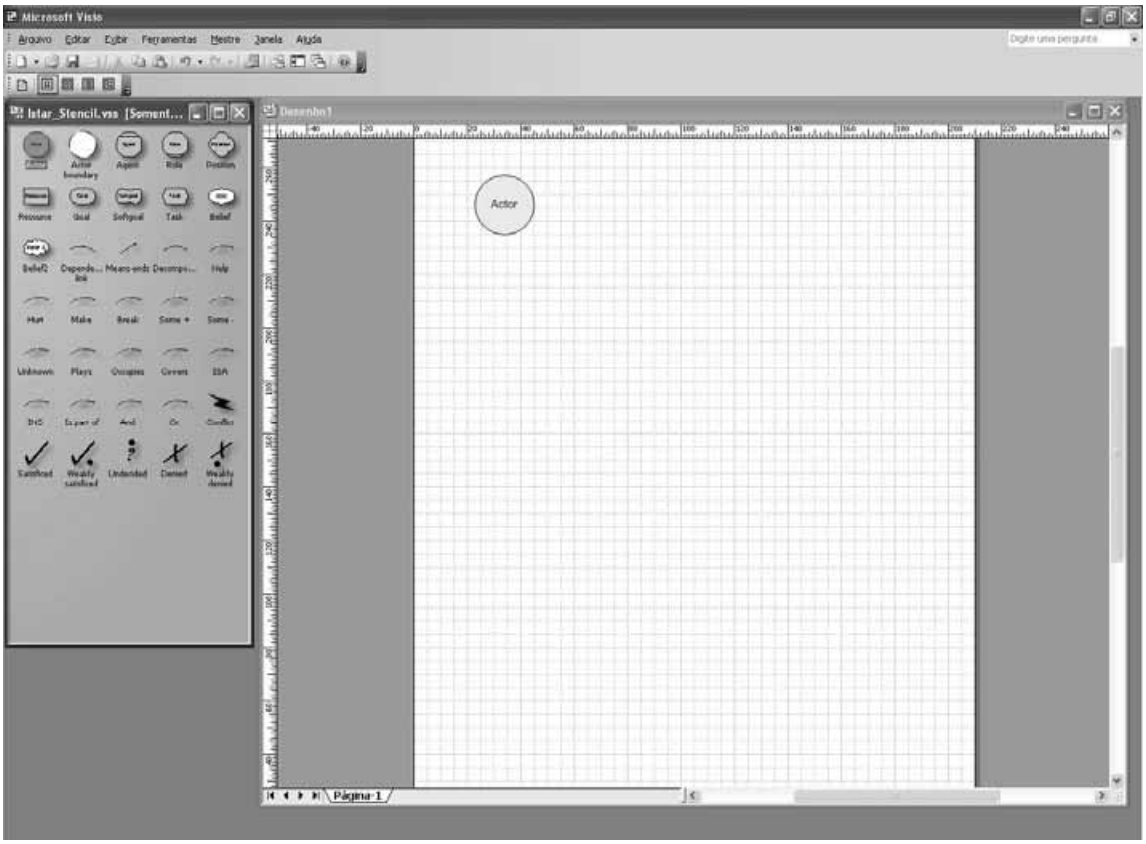


Figura 29 - Visio Template para desenvolvimento de modelos em i*.

3.2.11 TAOM4E

TAOM4E (*Tool for Agent-Oriented visual Modeling for the Eclipse Platform*) [Bertolini 2006][Susi 2005] é uma ferramenta desenvolvida em Java cujo propósito é prover suporte à modelagem e desenvolvimento orientado a agentes segundo a metodologia Tropos [Bresciani 2004]. Todo o seu desenvolvimento está de acordo com as recomendações da OMG (*The Object Management Group*) [OMG 2008] relativo ao uso de MDA (*Model Driven Architecture*).

Model Driven Architecure (MDA) [MDA 2008] é uma iniciativa da OMG que tem como objetivo prover uma maior abstração entre a especificação e a implementação de sistemas. Isto permite que a especificação, baseada em uma linguagem padrão de modelagem de sistemas - como a *Unified Modeling Language* (UML), por exemplo, se mantenha independente de plataforma e possa ser transformada em um modelo (código executável) de plataforma específica.

Na *Figura 30* é ilustrado o ambiente de modelagem TAOM4E. Apesar de a ferramenta prover suporte ao Tropos, esse *plug-in* não está de acordo com o *framework* i* original [Yu 1995] e também não está de acordo com a sua evolução, o i* segundo [i* Wiki 2008][Grau 2008]. Os modelo i* estão de acordo com uma variante [Bresciani 2004][Penserini 2006] do i* proposto por [Yu 1995]. Além disso, este *plug-in* dá suporte às atividades do Tropos versão da Itália ou Tropos'04 [Bresciani 2004], não atendendo às necessidades do grupo de pesquisa LER, como colocado nos objetivos descritos no capítulo de introdução deste trabalho - estar de acordo com o i* segundo [i* Wiki 2008][Grau 2008] e de acordo com o processo definido para o Tropos em [Silva, M. J. 2007].

Um outro ponto importante é que os modelos em i* gerados nesta ferramenta não estão de acordo com as melhores práticas de representação dos elementos. Por exemplo, os *links* de dependência não são representados com um 'D' apontando para seu destino de dependência como indicado em [i* Wiki 2008][Grau 2008] e sim como uma seta cheia (*Figura 30*).

Na *Figura 30* podemos observar, que assim como o OME (seção 3.2.3), o TAOM4E apresenta visões onde se pode trabalhar:

(i) Uma lista com os projetos e diretórios que estão sendo trabalhados (Navigator),

(ii) Uma visão geral do projeto que está sendo construído com as fases do Tropos e respectivos modelos (Outline),

(iii) Uma paleta com lista de opções - agrupadas por elementos e links - para construção dos modelos (Palette), e

(iv) Uma área de trabalho na qual o modelo é construído.

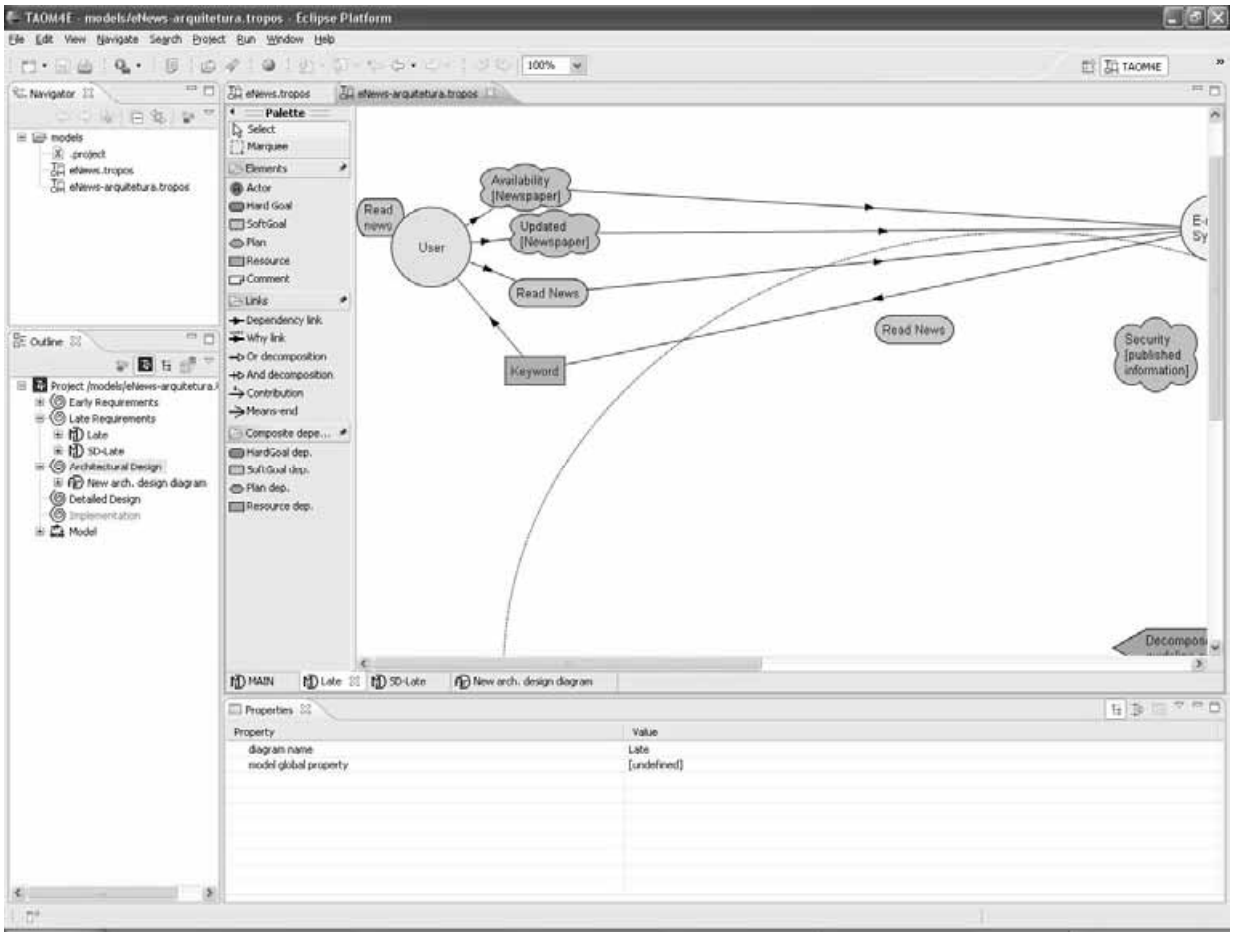


Figura 30 - Tela do TAOM4E.

3.3 Considerações

Hoje, o desenvolvimento orientado a agentes enfrenta algumas barreiras e desafios que o impedem de se tornar uma abordagem popular e difundida na comunidade. E uma das maiores carências que contribuem para este fator refere-se à ausência de metodologias sistemáticas e ferramentas de suporte que facilitem a especificação e estruturação de aplicações complexas como sistemas orientados a agentes [Castro 2006].

A metodologia Tropos oferece um *framework* que engloba as principais fases de desenvolvimento de software, com o apoio das seguintes atividades: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural e Projeto Detalhado. A partir da idéia inicial do projeto Tropos, duas abordagens diferentes surgiram: a versão de Toronto ou Tropos'01 [Castro 2002] e a versão da Itália ou Tropos'04 [Bresciani 2004]. Foi proposto em [Silva, M. J. 2007] um processo de desenvolvimento de software através da análise dessas duas versões do Tropos. Este processo compreende uma seqüência híbrida de atividades que reúne uma seleção de seqüências de atividades que permite aos usuários da metodologia utilizar as melhores características de cada abordagem. Portanto, nossa dissertação toma como referência o processo proposto no trabalho de [Silva, M. J. 2007].

As ferramentas apresentadas na seção 3.2 deste capítulo não estão completas, ou a modelagem com o i* não está de acordo com os guias e boas práticas apresentados em [i* Wiki 2008][Grau 2008],

ou elas não dão suporte ao processo do Tropos proposto segundo [Silva, M. J. 2007]. Além disso, estas ferramentas não possibilitam a checagem dos modelos gerados. Neste contexto, o uso de diferentes variações do i* através destas ferramentas produz modelos que provocam entendimentos divergentes - entre os envolvidos - a cerca dos sistemas modelados.

A tabela abaixo, é um quadro-resumo que apresenta os requisitos que indicamos como sendo importantes que uma ferramenta de modelagem i* deve ter. Para nosso estudo, daremos maior destaque a quatro requisitos: qual a versão do i* usada como referência, se a ferramenta permite a validação dos modelos gerados, qual a plataforma de desenvolvimento e linguagem utilizadas, e se o código é aberto permitindo uma fácil extensão.

Tabela 6 - Análise comparativa das ferramentas para modelagem i*.

	Versão i* framework	***Checagem de modelos***	***Suporte ao Tropos***	***Plataforma de desenvolvimento - Linguagem***	***Extensão***
GR-Tool	Tropos [Bresciani 2004] [Sannicoló 2002] [Tropos Project 2008]	Não	Não	Eclipse Platform, Java	Não
J-PRiM	i* original [Yu 1995]	Não	Não	Eclipse Platform, Java	Código aberto
OME	i* original [Yu 1995]	Não	Não	Eclipse Platform, Java	Através de plug-in
OpenOME	i* original [Yu 1995]	Não	Não	Eclipse Platform, Java	Através de plug-in
Redepend-React	i* original [Yu 1995]	Não	Não	VBA (Visual Basic for Applications)	Através de plug-in
SNet Tool	Criação de uma extensão para o i* [i* Wiki 2008]	Não	Não	Eclipse Prolog, Prolog	Não
Si* Tool	Tropos [Bresciani 2004] [Sannicoló 2002] [Tropos Project 2008] e Secure Tropos [Si* Tool 2006]	Não	Não	Java, Datalog	Não
T-Tool	Formal Tropos [T-Tool 2003]	Não	Não	Perl	Código aberto
DesCARTES	Tropos [Bresciani 2004] [Sannicoló 2002] [Tropos Project 2008] e i* original [Yu 1995]	Não	Versão Toronto ou Tropos'01 [Castro 2002]	Eclipse Platform, Java (usa EMF e GEF)	Através de plug-in
Visio Template	Não segue nenhuma versão, é customizável	Não	Não	Extensão do Visio (proprietário)	Extensão do template
TAOM4E	Tropos [Bresciani 2004] [Sannicoló 2002] [Tropos Project 2008]	Não	Versão Itália ou Tropos'04 [Bresciani 2004]	Eclipse Platform, Java (usa EMF e GEF)	Através de plug-in

Portanto, a intenção do nosso trabalho nos próximos capítulos será a de desenvolver uma ferramenta para a construção de modelos i* que atenda aos requisitos buscados nas ferramentas apresentadas no quadro-resumo acima. Para tanto, em nosso estudo iremos focar no i* apresentado [i* Wiki 2008][Grau 2008], pois este projeto foi criado com a intenção de permitir o trabalho colaborativo a cerca do i *, possibilitando aos usuários desta linguagem de modelagem uma visão comum de seu uso. Esse entendimento comum é importante para que todos tenham a mesma visão dos sistemas modelados utilizando o i*, não produzindo assim um sistema em desacordo como o ambiente. Durante este desenvolvimento, estaremos preocupados em projetar uma ferramenta que permita a validação dos modelos gerados segundo as restrições e *guidelines* definidas em [i* Wiki 2008][Grau 2008].

Capítulo

4 IStar Tool - Uma Ferramenta de Apoio à Modelagem de Requisitos Organizacionais em i*

Este capítulo trata da criação de um meta-modelo para a nova versão do i [*i* Wiki 2008*][Grau 2008]. Primeiro, será apresentado o processo a ser seguido - utilizando-se o framework GMF do Eclipse - para o desenvolvimento de um editor no qual se possa realizar a modelagem segundo o meta-modelo do i* a ser criado. Em seguida, apresentaremos todas as restrições e indicações de boas práticas para construção dos modelos i*. Por fim, será apresentada a construção dos modelos necessários na implementação da ferramenta proposta.*

4.1 Introdução

Um modelo consiste de um conjunto de elementos, e pode ser usado como uma representação simbólica de algo físico, abstrato ou da realidade. A idéia de criação e implementação de algo a partir de modelos é a base do Desenvolvimento Orientado a Modelos, ou MDD (*Model Driven Development*) [MDA 2008]. O uso dessa abordagem vem ganhando cada vez mais aceitação na indústria e no meio acadêmico, pois dentre outros fatores facilita o projeto em todos os níveis arquiteturais e permite a análise do que foi gerado incluindo validação e verificação. Além disso, seu uso é enfatizado na proposta da OMG de Arquitetura Baseada em Modelos (MDA - *Model Driven Architecture*) [MDA 2008].

O principal motivo para a escolha dessa abordagem é a simplicidade e agilidade proporcionada pela mesma. Neste sentido, para o desenvolvimento da ferramenta proposta, adotaremos como tecnologia orientada a modelos o *framework* GMF (*Graphical Modelling Framework*) [GMF 2008], que permite desde a modelagem do domínio até a geração automática de código para representação e edição gráficas dos objetos presentes nos modelos. Na seção a seguir, apresentamos o processo deste *framework* e sua aplicação no desenvolvimento da ferramenta proposta.

4.2 GMF Framework

O *framework* de modelagem gráfica GMF (*Graphical Modelling Framework*) [GMF 2008] é um projeto da *Eclipse Foundation* [Eclipse 2008]. Seu propósito é fornecer uma infra-estrutura para o desenvolvimento de editores gráficos baseados nos *frameworks* – também do Eclipse – EMF [EMF 2008], que auxilia a especificação de meta-modelos e provê funcionalidades para a geração automática do código Java respectivo, e GEF [GEF 2008] utilizado para a criação de editores gráficos genéricos.

A *Figura 31* ilustra o ambiente onde um editor construído segundo o GMF opera: ambiente de execução do GMF e componentes disponibilizados pelo EMF e GEF, todos estes executando usando a plataforma Eclipse.

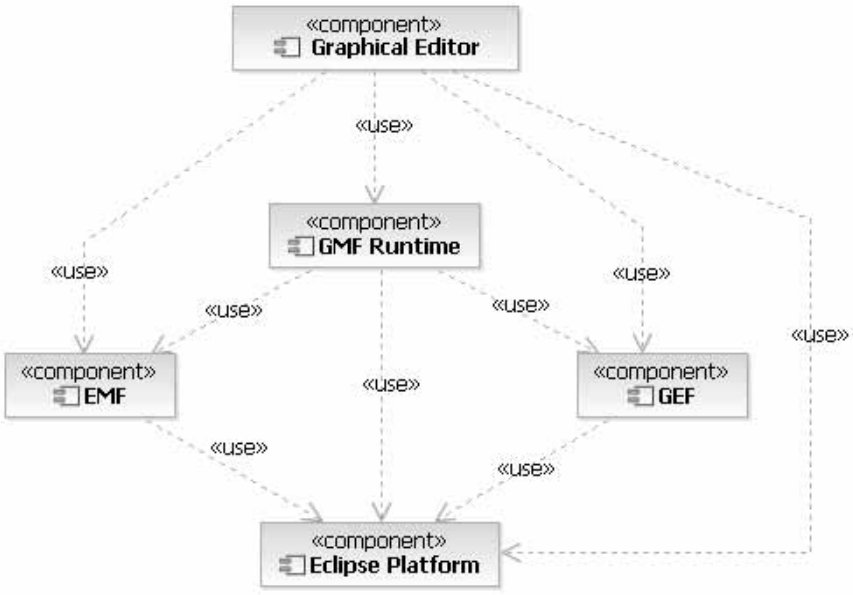


Figura 31 - Framework GMF e dependências [Venkatesan 2006].

GMF [GMF 2008] é um *plug-in* para o Eclipse, que fornece uma API[3] e um conjunto de ferramentas para a produção automática de editores (baseados em modelos EMF) aplicados a um determinado domínio. Este *framework* de desenvolvimento é dividido na construção de quatro modelos que são utilizados para representar diferentes aspectos de um editor. Estes modelos são: Modelo Gráfico (*Graphical Model*), Modelo Ferramental (*Tools Model*), Modelo de Mapeamento (*Mapping Model*) e o Modelo Gerador (*Generator Model*). Cada um desses modelos é composto de classes e associações que representam algum elemento do diagrama que, quando processado, será traduzido em código GEF que implementa o editor.

Para a construção do editor alguns passos devem ser seguidos utilizando-se o *framework* GMF (vide *Figura 32*). Esse processo é iniciado criando-se um projeto GMF. Neste projeto deverão ser criados: o Modelo de Domínio (definição de um meta-modelo para o diagrama que será possível construir com o editor - extensão .ecore), a Definição Gráfica (ou Modelo Gráfico - extensão .gmfgraph), e a Definição Ferramental (ou Modelo Ferramental - extensão .gmftool). Esses três modelos formam a base para a criação do editor, é a partir deles que o Modelo de Mapeamento (extensão .gmfmap) é criado, ligando assim os elementos do domínio, aos elementos gráficos, e aos elementos ferramentais. O último modelo, o Modelo de Geração (ou gerador), é derivado a partir do modelo de mapeamento e usado para gerar o código executável do editor gráfico.

O diagrama abaixo ilustra os principais componentes e modelos usados durante o desenvolvimento baseado em GMF.

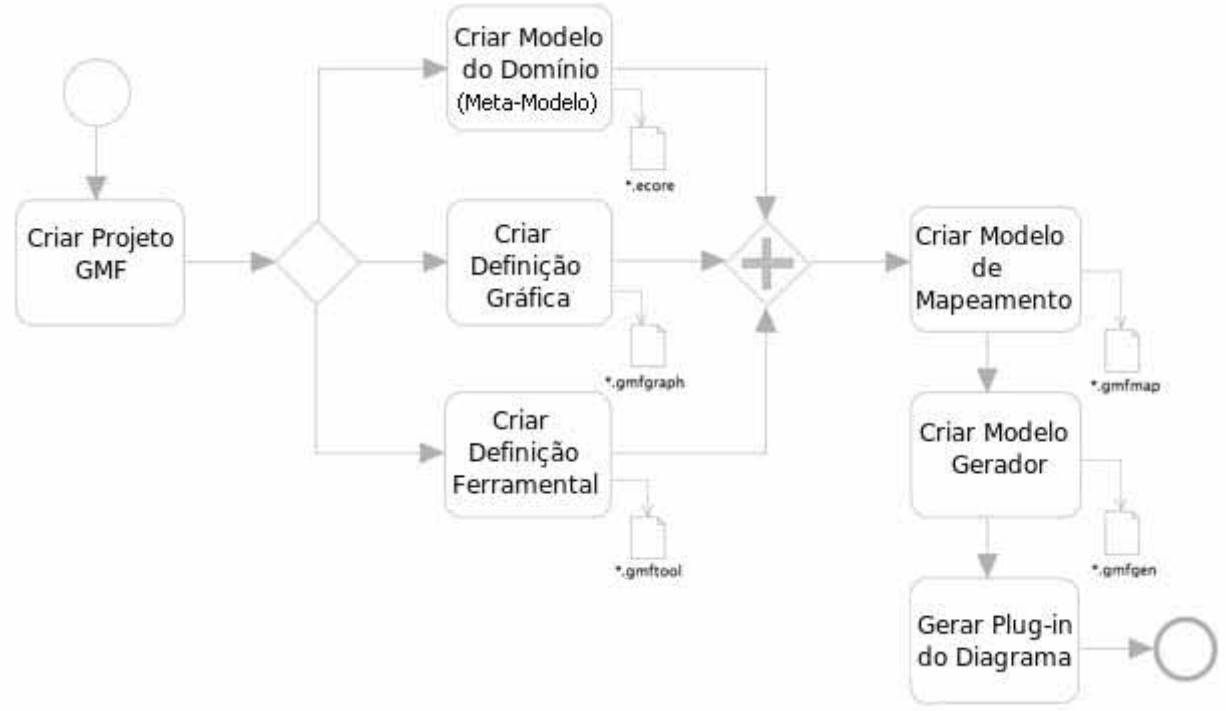


Figura 32 - Visão Geral do processo para criação de um editor gráfico com GMF [GMF 2008].

[3] API, *Application Programming Interface* (ou Interface de Programação de Aplicativos) é um conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades por programas aplicativos. De modo geral, a API é composta por uma série de funções acessíveis somente por programação, e que permitem utilizar características do software menos evidentes ao usuário tradicional.

O núcleo do desenvolvimento de um projeto GMF é a definição do modelo gráfico. Este modelo contém informação relacionada apenas aos elementos gráficos que serão visualizados no editor em construção, porém este modelo não tem associação direta com o meta-modelo para o qual ele irá fornecer representação. Isso se deve ao fato de que um dos objetivos do GMF é permitir que a definição gráfica possa ser reusada para diferentes domínios. Para isto, como pode ser visto no fluxo representado pela *Figura 32*, são necessários diferentes modelos de mapeamento para fazer a associação das definições gráficas e ferramentais para os modelos de domínio desejados.

Uma vez que os mapeamentos foram definidos apropriadamente, o *framework* GMF fornece um modelo gerador para permitir que alguns detalhes necessários de implementação sejam definidos. O Modelo Gerador terá como alvo um modelo final, este modelo é o código executável do editor gráfico. Esse código fará a ligação entre a notação e o(s) modelo(s) de domínio quando o usuário estiver trabalhando com um diagrama, e também fornecerá a persistência e sincronização entre eles. Um aspecto importante deste ambiente de execução é que ele fornece uma abordagem baseada em serviços para funcionalidades EMF e GEF, e é capaz de ser aproveitado para outras aplicações existentes.

Nas seções abaixo a definição de cada modelo neste fluxo de desenvolvimento utilizando o GMF é apresentada.

4.2.1 Definição do Modelo de Domínio

O meta-modelo de um diagrama (ou Modelo do Domínio) é definido segundo o *framework* EMF. É através deste que é indicada a forma em que os nós do diagrama podem interagir uns com os outros. Este meta-modelo em formato XMI [XMI 2007] é conhecido como modelo Ecore, um padrão através do qual é possível definir um meta-modelo em diferentes ferramentas, tal como Rational RSA [IBM RSA 2007]. O uso do formato XMI (ou XML *Metadata Interchange*) se justifica por este facilitar a troca de meta-dados entre as ferramentas de modelagem. XMI é um padrão da OMG (*Object Management Group*) [OMG 2008] para troca de informações baseado em XML.

No Eclipse há três formas de se criar este modelo [Eclipsepedia 2008]: editando diretamente um arquivo XMI, que pode ser útil quando é necessário copiar grandes blocos de definição entre modelos; editando o modelo através da árvore de definição de um editor EMF, que fornece uma abordagem metódica e não ambígua; ou usando o editor GMF que permite que o modelo seja criado usando um diagrama semelhante a um diagrama de classes.

Na definição de um meta-modelo os nós de um diagrama são especificados como classes (EClasses), que podem ter como propriedades um número de atributos (EAttributes), referências (EReferences) e métodos (EOperations) [Eclipsepedia 2008].

Os atributos (EAttributes) são usados para definir os atributos que aparecerão em cada nó do diagrama, devendo ser conferido a cada atributo um nome e um tipo. Os tipos dos atributos podem ser qualquer um do padrão Java, tais como String, Long ou Boolean. O nome do atributo não é restrito, mas é uma boa prática nomeá-lo de forma que ele possa ser diferenciado de outros atributos em classes diferentes presentes no modelo em definição.

As referências (EReferences) são atributos usados para definir uma relação entre classes ou entre uma classe e ela mesma. Configurar uma referência entre duas classes permite que estas sejam conectadas no editor gráfico que se pretende construir. Uma relação entre classes também pode ser modelada através de uma outra classe, esta, portanto precisa ter duas EReferences, uma para o nó destino e outra para o nó origem. EReferences tem diversas propriedades que precisam ser configuradas a fim de que estas trabalhem adequadamente. O nome da referência deve ser configurado e é indicada como uma boa prática dar um nome que o identifique unicamente dentro do meta-modelo. Isto irá facilitar a identificação da relação nos estágios mais avançados do processo de criação do editor gráfico. Além do nome, outras propriedades devem ser configuradas para uma referência.

A propriedade que identifica o tipo da referência (eType) deve ser configurado de acordo com a classe da origem e do destino. Os limites superior e inferior (*upper bound* e *lower bound*) da referência podem ambos ser configurados a fim de restringir a cardinalidade do relacionamento. Definindo o limite superior para -1, por exemplo, permite uma relação de 1 para n. Para cada referência criada a propriedade chamada *containment* deve ser configurada para *true* ou *false*. Essa configuração quando definida para *true*, permitirá que o nó alvo da relação seja incluído como um nó filho na referência. Isto é útil quando se deseja modelar um comportamento recursivo das relações.

As operações (EOperations) são usadas para definir algum comportamento (operações) de uma classe no meta-modelo, comportamento esse que não pode ser determinado unicamente pelo projeto do meta-modelo. Existem várias formas nas quais EOperations podem ser usadas, tal como na criação de um método em Java para classe onde está sendo definida a operação. O *framework* EMF automaticamente adiciona métodos (código implementado) para toda EOperations definida na classe do meta-modelo. Um exemplo de uso desta propriedade de um nó (objeto) no modelo é que através da definição de EOperations é possível definir restrições em GMF.

Um exemplo de definição de um meta-modelo em Ecore encontra-se na *Figura 33*. Utilizaremos como exemplo a construção de um editor para a criação de *orgcharts* utilizados para representar a estrutura hierárquica de uma organização representando os empregados da organização e seus respectivos gerentes.

Nesta figura damos destaque ao painel de trabalho do *framework* GMF - *GMF Dashboard*. É através desta área que os modelos são criados, definidos, combinados e transformados a fim de gerar o editor gráfico.

Este mesmo modelo também pode ser visualizado como um diagrama semelhante a um diagrama de classes, como pode ser visualizado na *Figura 34*.

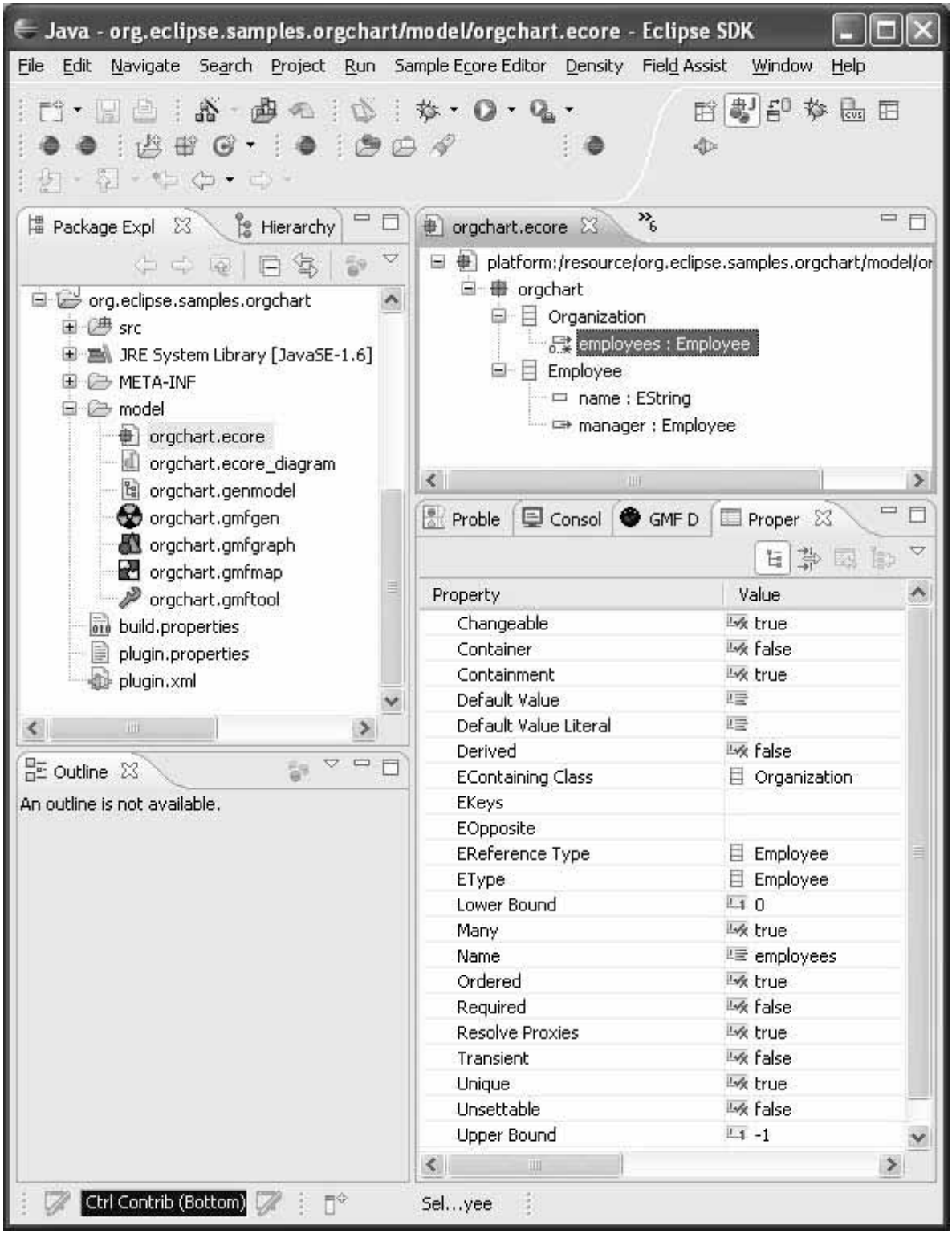


Figura 33 - Exemplo de modelo Ecore - Orgchart.

Uma vez que este meta-modelo tenha sido construído, um modelo chamado EMF GenModel deve ser criado a partir do arquivo Ecore. É através da construção deste modelo que o código implementado do modelo é gerado (*Model Code* e *Edit Code*), clicando-se com o botão direito no modelo EMF GenModel através das opções *Generate Model Code* e *Generate Edit Code*. Esse código gerado nada mais é do que as classes, em linguagem Java, que representam os elementos de domínio definidos. A

opção *Generate Model Code* gera as classes básicas para cada elemento dentro da pasta *src* no mesmo projeto onde o meta-modelo foi criado - *org.eclipse.samples.orgchart*. E a opção *Generate Edit Code* gera as classes provedoras de acesso às propriedades de cada elemento dentro do projeto *org.eclipse.samples.orgchart.edit*.

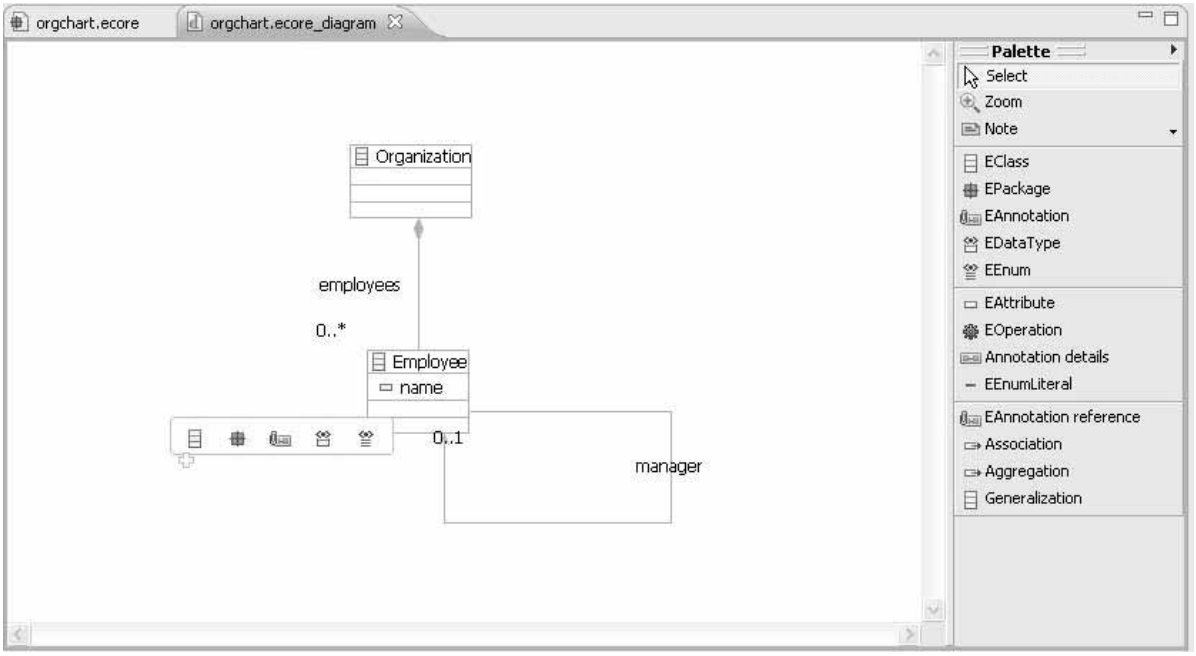


Figura 34 - Exemplo de diagrama Ecore - Orgchart.

4.2.2 Definição do Modelo Gráfico

O modelo gráfico é definido através do *framework* GMF com o propósito de definir os elementos gráficos de um diagrama. Isto inclui a definição dos nós, *labels* (rótulos), e *links* de relacionamento. Este modelo contém: uma galeria de figuras (*Figure Gallery*) onde estão as figuras que serão utilizadas para definir as formas a serem utilizadas pelos elementos; e os elementos que definem os nós, os *labels* e os relacionamentos (ligações).

A *Figura 35* dá um exemplo de como esse modelo gráfico é construído. Podemos visualizar neste exemplo que foram criadas para a galeria de figuras: uma figura para representar um nó (*Figure Descriptor EmployeeFigure*), uma linha para representar a conexão entre os nós (*Figure Descriptor EmployeeManagerFigure*) e um elemento para representar a decoração a ser utilizada no destino da linha de conexão (*Polyline Decoration EmployeeManagerTargetDecoration*). Além das figuras, foram criadas também neste exemplo o nó (*Node Employee*) que utilizará a figura definida na galeria como um nó (*EmployeeFigure*), a conexão (*Connection EmployeeManager*) que será utilizada para ligar os nós e usa a figura definida na galeria como uma linha (*EmployeeManagerFigure*), e o *label* que será utilizado para dar nome aos nós (*Diagram Label EmployeeName*).

GMF fornece um *wizard* para a criação de Modelos Gráficos. Este assistente permite que os nós, *labels* e conexões desejados sejam selecionados, e automaticamente um conjunto de padrões são fornecidos. A saída do assistente é um Modelo Gráfico simples, que contém as figuras e os elementos

criados para as classes selecionadas, *labels* e conexões. Este modelo gerado pode ser modificado a fim de produzir o comportamento desejado.

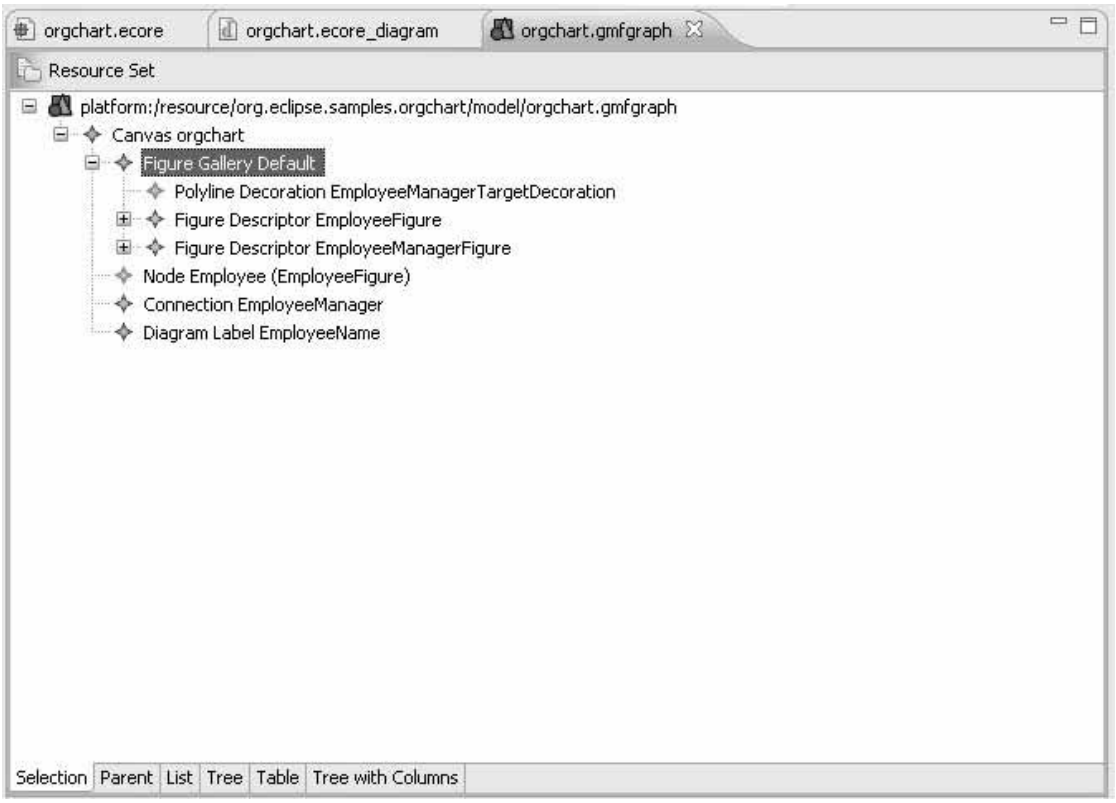


Figura 35 - Exemplo de definição de modelo gráfico - Orgchart.

Existem três grandes modificações que podem ser realizadas no mapeamento gráfico. Primeiro, há a possibilidade de modificação da representação gráfica padrão dos nós, gerada pelo *wizard* do GMF. Isto é realizado pela definição de novas figuras na Galeria de Figuras e pela configuração apropriada dos nós para que estes usem as novas figuras. Estas novas figuras podem ser uma das figuras padrões disponíveis, tais como retângulos (*Rectangles*), elipses (*Ellipses*) e polígonos (*Polygons*), ou podem ser figuras customizadas que requer implementação de código (*Custom Figures*). Para usar uma figura customizada, é necessário definir as formas e como estas serão preenchidas - através dos métodos *fillShape()* e *outlineShape()*.

A segunda modificação que pode ser realizada sobre o modelo gráfico gerado pelo assistente é a adição de compartimentos (*Compartment*) que podem ser usados no modelo de mapeamento. Este é um processo que possui dois estágios: (i) o compartimento deve ser adicionado na raiz do modelo gráfico e (ii) a figura a ser utilizada por este deverá ser definida. Os compartimentos devem ter um nome, e podem opcionalmente, serem forçados a exibir seu nome no editor gráfico que está sendo construído.

A terceira modificação que pode ser aplicada é a alteração do estilo das conexões. Existem duas opções: (i) modificar o estilo da linha, (ii) ou modificar o estilo das extremidades que aparecem na origem e no destino da conexão. A linha é modifica através de propriedades no próprio modelo gráfico. Algumas das propriedades que podem ser alteradas são: tipo da linha, largura da linha, e os elementos gráficos a serem utilizados nas extremidades. O estilo, a ser utilizado na origem e no destino, envolve a

criação de uma Decoração Customizada (*Custom Decoration*), semelhante ao que é feito quando são criadas figuras customizadas.

Abaixo, a *Figura 36* explana como a definição gráfica de um diagrama é vista [Eclipsepedia 2008]. Em resumo, um diagrama que está sendo modelado graficamente contém um conjunto de elementos, que podem ser nós ou conexões. Cada elemento do diagrama é representado por uma figura descrita através do Descritor de Figuras - este descritor contém o conjunto de figuras que irão compor o elemento.

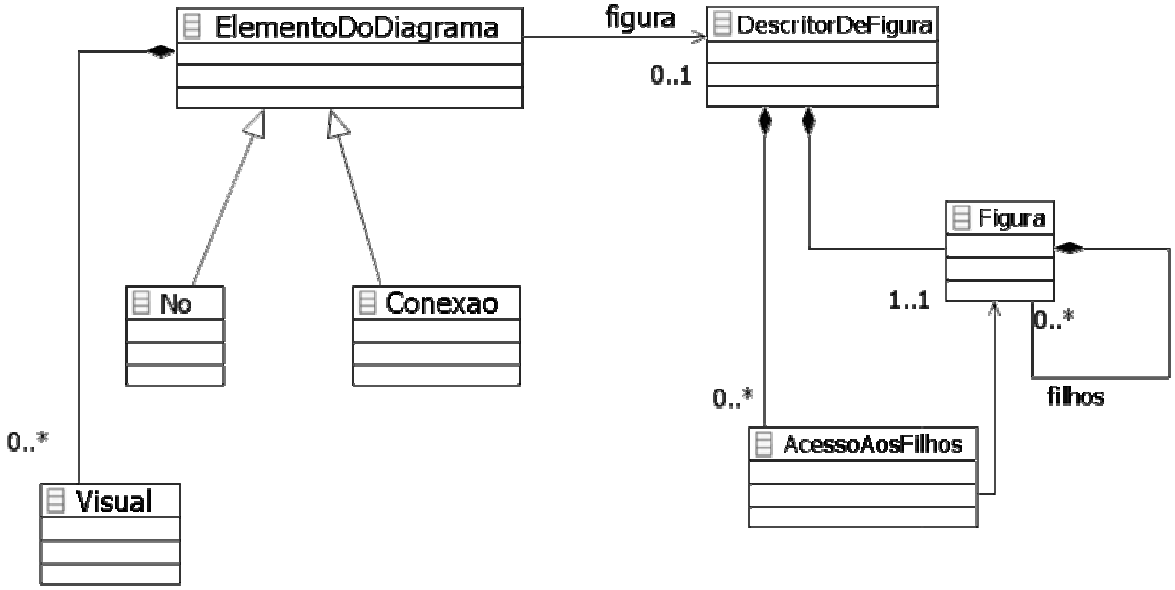


Figura 36 - Visão de construção de modelos gráficos - gmfgraphs.

4.2.3 Definição do Modelo Ferramental

O Modelo Ferramental é utilizado para definir o *toolbar* e os menus a serem usados com o editor gráfico que está sendo criado. Este modelo permite que diversos tipos de menu sejam incluídos, tais como menus de contexto (*Context*), pop-up e menus principais. O *toolbar*, que é o principal foco deste modelo, é definido dentro da paleta (*Palette*) e contém um Grupo de Ferramentas (*Tool Groups*), contendo a definição das ferramentas (*Tools*) a serem exibidas na paleta.

Na *Figura 37*, temos um exemplo da estrutura de um modelo ferramental gerado. Este modelo apresenta um grupo de ferramentas chamada *orgchart* que contém as seguintes opções de ferramenta: *Employee* e *EmployeeManager*. As opções de ferramentas, como pode ser observado na figura são criadas pelo elemento *Creation Tool*.

Aqui o GMF também fornece um assistente para gerar um modelo ferramental automaticamente. Este produz um conjunto padrão que pode ser modificado a fim de que este modelo forneça o comportamento desejado. A modificação mais comum que pode ser realizada em cima do modelo gerado é dividir os grupo de ferramentas em categorias diferentes e mudar os nomes das opções de ferramentas criadas.

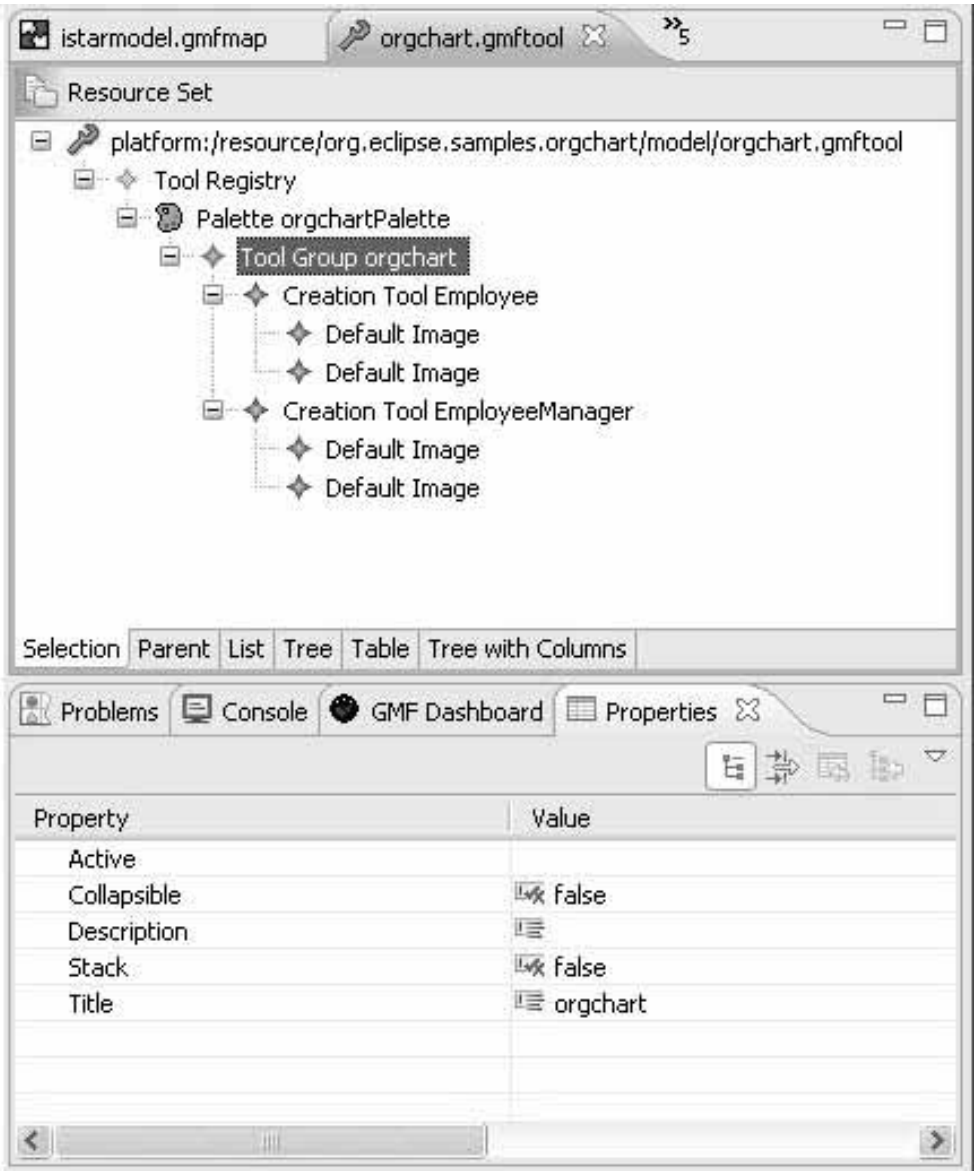


Figura 37 - Exemplo de definição de modelo ferramental - Orgchart.

4.2.4 Criação do Modelo de Mapeamento

O Modelo de Mapeamento é usado no GMF para mapear os modelos gráfico e ferramental com o meta-modelo. Este modelo pode ser automaticamente gerado através de um *wizard*, que o GMF disponibiliza, mas este é pouco confiável e, portanto o modelo de mapeamento resultante deve ser verificado. O mapeamento é criado definindo-se as referências para os nós chamadas *Top Node Reference* e as referências para as conexões chamadas *Link Mapping*.

Cada referência a um nó contém um nó de mapeamento (*Node Mapping*) que são usados para mapear: (i) os nós no diagrama ao elemento definido no meta-modelo, (ii) os nós no diagrama aos elementos gráficos definidos no modelo gráfico, (iii) e os nós no diagrama à ferramenta de criação definida no modelo ferramental. Cada *Node Mapping* criado contém outros mapeamentos, onde rótulos (*Label Mappings*), referências a elementos filhos (*Child References*), e compartimentos podem ser definidos (*Compartment Mappings*). O mapeamento dos rótulos referencia um rótulo do diagrama no

modelo gráfico a um ou mais atributos no meta-modelo. O mapeamento de compartimento associa um compartimento do modelo gráfico a uma classe definida no meta-modelo.

Cada referência a uma ligação contém um conjunto de propriedades a serem definidas que envolvem: (i) o mapeamento do elemento no meta-modelo que representa a ligação na propriedade *Domain meta information - Source Feature* e *Target Feature*, (ii) o mapeamento do elemento gráfico que representa a ligação na propriedade *Visual representation - Diagram Link*, e (iii) o mapeamento do elemento ferramental que representa a ligação na propriedade *Visual representation - Tool*.

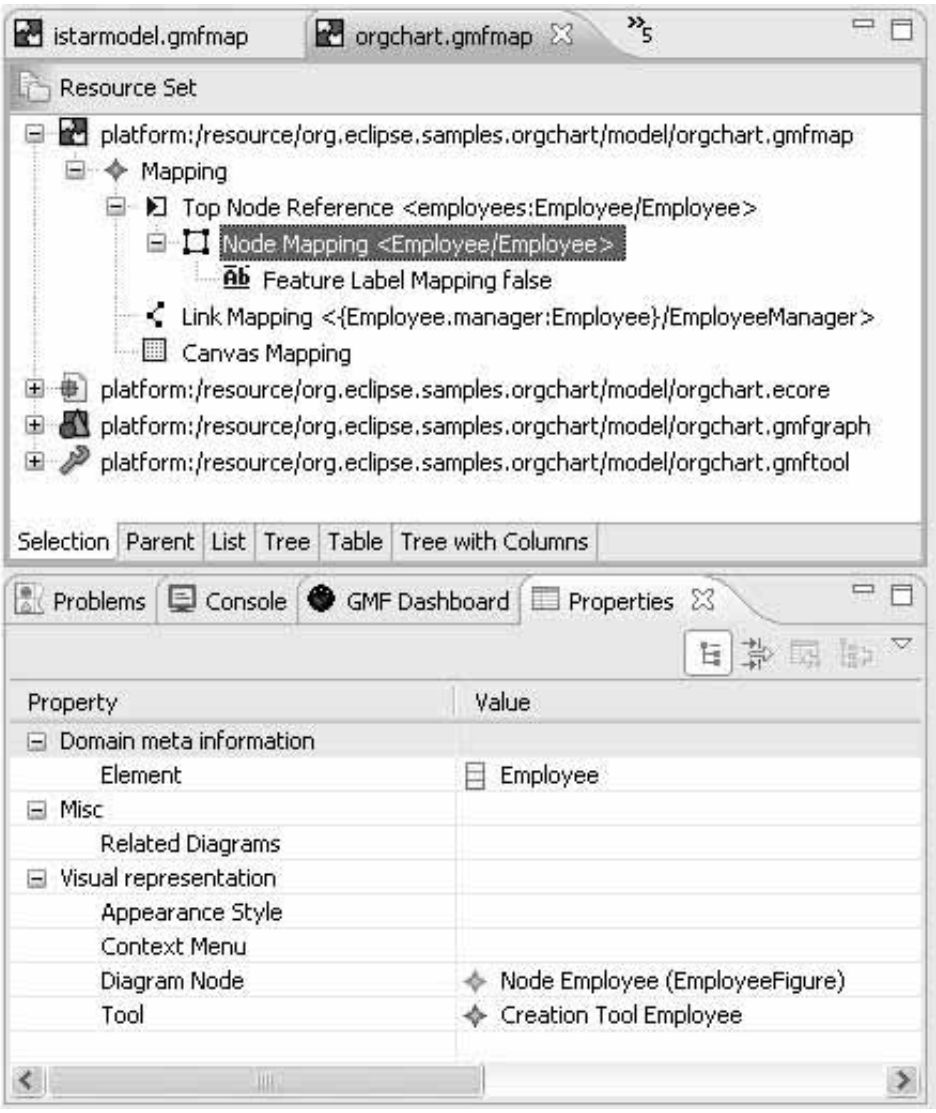


Figura 38 - Exemplo de definição de modelo de mapeamento - Orgchart.

Na *Figura 38* podemos visualizar o modelo de mapeamento do nosso exemplo. Nele temos:

(i) O mapeamento do nó *Employee* (*Node Mapping*) para: o elemento definido no meta-modelo (propriedade *Domain meta information - Element : Employee*), para o nó definido no modelo gráfico (propriedade *Visual representation - Diagram Node : NodeEmployee(EmployeeFigure)*) e para o elemento definido no modelo ferramental (propriedade *Visual representation - Tool : Creation Tool Employee*).

(ii) O mapeamento do rótulo (*Feature Label Mapping*) a ser usado pelo nó *Employee* para: o atributo definido no meta-modelo (propriedade *Features*) e para o rótulo definido no modelo gráfico (propriedade *Diagram Label*).

(iii) O mapeamento da ligação *EmployeeManager* (*Link Mapping*) entre nós. A ligação em nosso exemplo é mapeada para: o elemento definido no meta-modelo (atributo *manager*), para a conexão criada no modelo gráfico (propriedade *Diagram Link*), e para o elemento definido no modelo ferramental (propriedade propriedade *Visual representation - Tool*).

É interessante observar que como já apresentado nas seções anteriores, o modelo de mapeamento usa os modelos de domínio (ou meta-modelo), gráfico e ferramental, e por isso uma referência a esses modelos aparece no modelo de mapeamento.

Uma vez que o Modelo de Mapeamento tenha sido finalizado, ele é usado para originar o modelo que irá gerar o código executável do editor gráfico modelado até o momento.

4.2.5 Criação do Modelo de Geração

O Modelo de Geração é usado no GMF para produzir o código executável do diagrama. Tal modelo contém propriedades que podem ser modificadas a fim de produzir características específicas ao código. Por exemplo, às figuras definidas no Modelo Gráfico pode ser configurado um tamanho padrão.

O modelo é resultado de uma transformação que o *framework* GMF aplica sobre o modelo de mapeamento. O modelo de geração normalmente não é modificado, contudo uma modificação provável é a modificação de localização de pacotes onde o código do diagrama é gerado. Essa mudança se faz necessária se existirem muitos diagramas sendo produzidos de um único meta-modelo, para evitar que o código de um diagrama sobrescreva outro.

Um outro fator importante no uso do GMF é a possibilidade de adição de restrições ao comportamento do diagrama. Tais restrições são definidas em OCL (*Object Constraint Language*) [OCL 2007] no modelo de mapeamento e podem ser utilizadas para restringir algumas ações (como restringir a conexão entre dois elementos do diagrama), ou para realizar validação de alguns aspectos do modelo após sua construção - utilizando o *framework* de validação fornecido pelo EMF.

OCL é uma linguagem declarativa, que utiliza palavras comuns para expressar operações realizadas sobre um modelo. E através desta é possível descrever as regras que se aplicam aos modelos de classes criados.

Um exemplo do modelo de geração produzido pode ser visualizado na *Figura 39*. Na figura é mostrada o *GMF Dashboard*, uma visão fornecida pelo *framework* GMF para auxiliar a atividade de criação/transformação dos modelos, e pode ser visualizado também o modelo gerador criado a partir do modelo de mapeamento. No modelo gerador criado observamos que este faz referência aos modelos ecore e genmodel, e a sua estrutura em árvore (*Gen Diagram Generator*) reúne todas as definições provenientes do modelo de mapeamento.

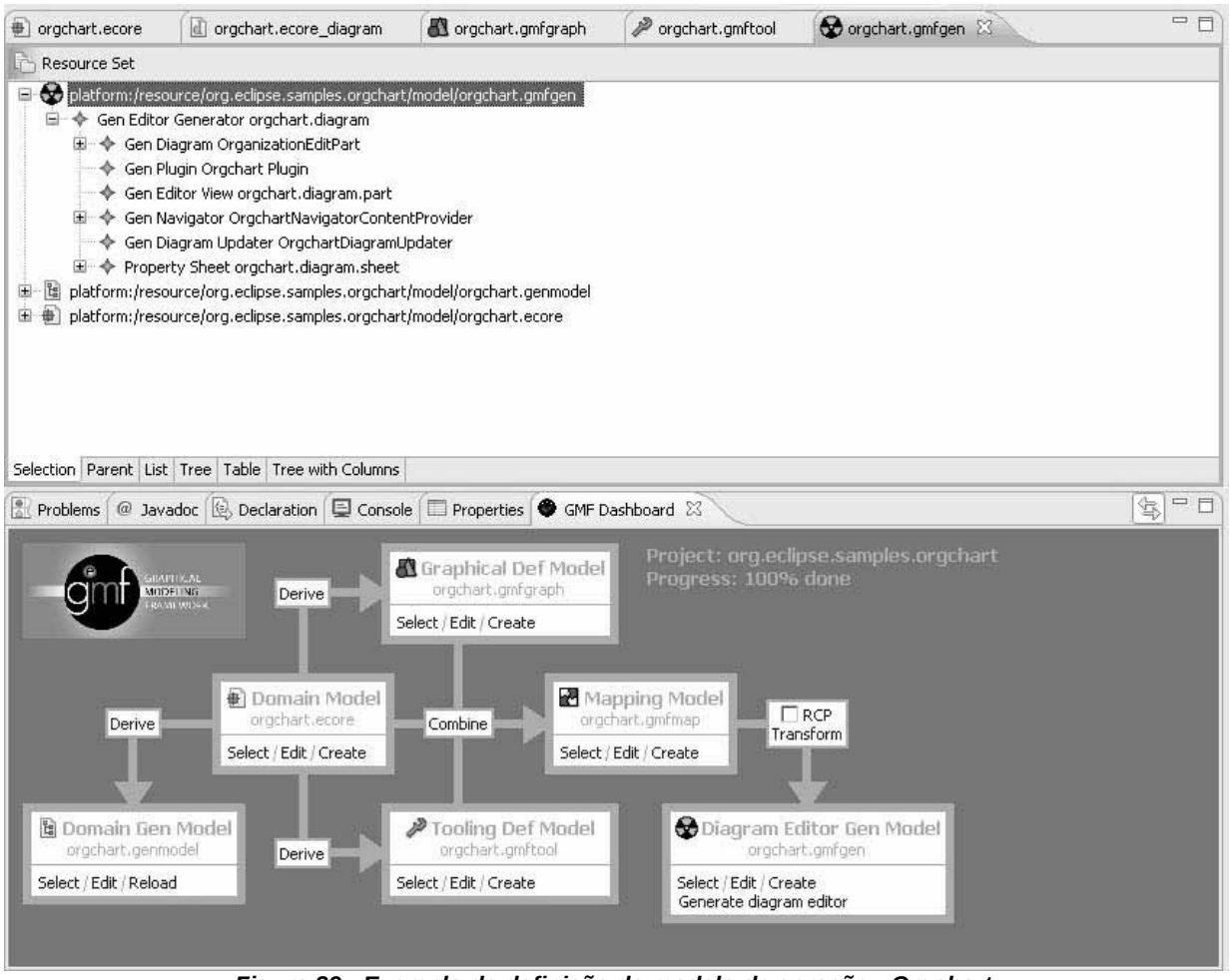


Figura 39 - Exemplo de definição de modelo de geração - Orgchart.

*4.3 Uso do Framework GMF na Construção da Ferramenta de Modelagem i**

Como apresentado nos capítulos anteriores, a linguagem de modelagem i* é utilizada nas fases de requisitos inicial e final da metodologia Tropos para a produção de dois artefatos de saída: modelo de atores e modelo de Objetivos - em outras palavras, os modelos SD (*Strategic Dependecy Model*) e SR (*Strategic Rationale Model*).

Esta seção apresentará, portanto os modelos desenvolvidos - segundo o *framework* GMF [GMF 2008], para representar diagramas em i* [i* Wiki 2008][Grau 2008]. Antes, porém, apresentaremos as restrições e boas práticas inerentes à criação dos diagramas i* segundo o [i* Wiki 2008][Grau 2008] e que serão consideradas na construção dos modelos que compõem a ferramenta de modelagem i*.

Observamos que tanto a modelagem quanto a implementação da ferramenta foram todas geradas em inglês, pois sendo esta uma linguagem universal, isso irá permitir que a ferramenta desenvolvida seja utilizada (e até mesmo, posteriormente estendida) não só pelos membros do grupo de pesquisa LER, mas por qualquer outro pesquisador da comunidade do i* e do Tropos que tenha interesse em fazê-lo.

4.3.1 Restrições e Guidelines

Para cada modelo i* desenvolvido, algumas restrições e boas práticas [i* Wiki 2008][Grau 2008] quanto à sua construção devem ser consideradas. Abaixo serão listados os guias e boas práticas considerados em [i* Wiki 2008][Grau 2008] organizados pelos modelos que podem ser gerados utilizando-se o i*: Modelo de Dependência Estratégica (SD) e Modelo Estratégico da Razão (SR).

Modelo SD

Como já apresentado, o modelo SD é composto por um conjunto de nós e ligações, onde cada nó representa um ator e cada *link* entre dois atores indica que um depende do outro para algo, a fim de que o ator dependente atinja alguns objetivos. O modelo SD é usado para expressar uma rede de relações intencionais e estratégicas entre atores. Tal modelo retrata a dependência estratégica entre atores, mas não retrata as razões por detrás dessas dependências, que só serão contempladas no modelo SR.

Guideline 1. Um ator não deve ter dependência com ele mesmo.

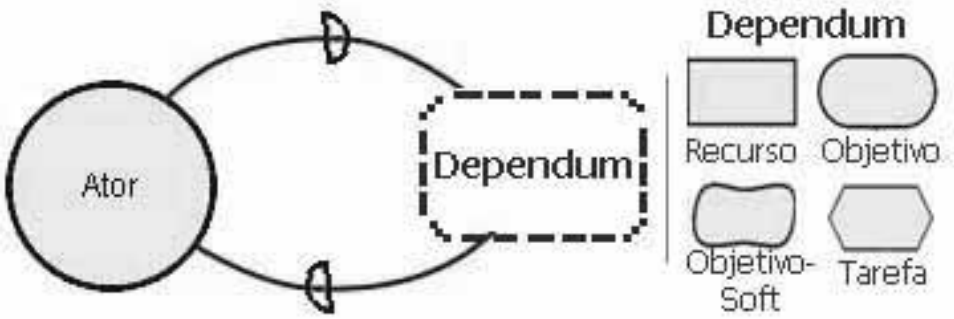


Figura 40 - Guideline 1: um ator não deve ter dependência com ele mesmo.

Guideline 2. Não deve existir um ator, ou um elemento de dependência, sem ligação.

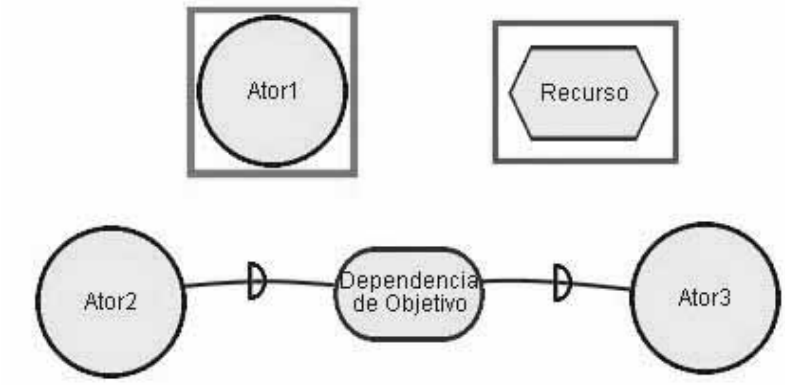


Figura 41 - Guideline 2: não deve exisitir um ator ou um elemento de dependência sem ligação.

Guideline 3. A associação entre atores '*IS A'* e '*Is Part Of*' só deve ocorrer entre atores do mesmo tipo.

Por exemplo: A ligação entre o Agente1 e o Ator1 através de uma associação *ISA* está errada.

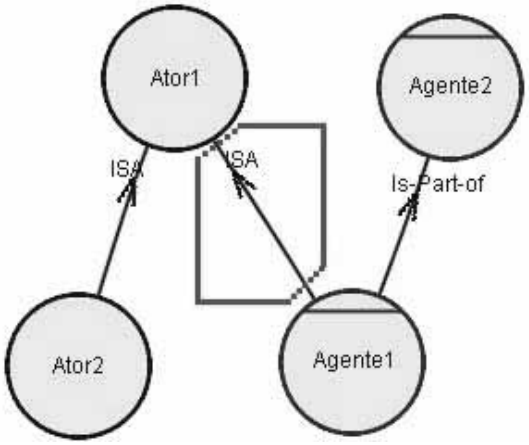


Figura 42 - Guideline 3: não deve exisitir atores de diferentes tipos em uma associação Is-Part-Of ou ISA.

Guideline 4. A associação entre atores '*Plays*' só deve ocorrer de um Agente para um Papel.

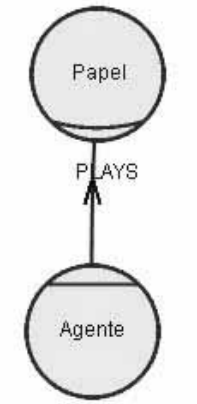


Figura 43 - Guideline 4: associação Plays somente entre um Papel e um Agente.

Guideline 5. A associação entre atores '*Covers*' só deve ocorrer de uma Posição para um Papel.

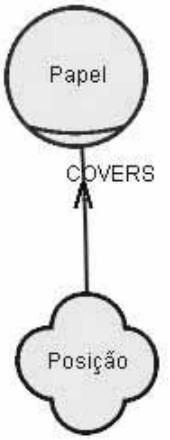


Figura 44 - Guideline 5: associação Covers somente entre uma Posição e um Papel.

Guideline 6. A associação entre atores '*Occupies*' só deve ocorrer de um Agente para uma Posição.

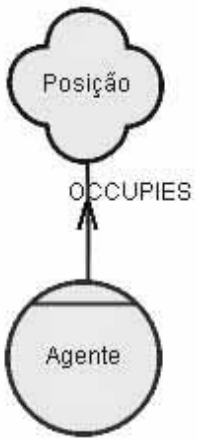


Figura 45 - Guideline 6: associação Occupies somente entre uma Posição e um Agente.

Guideline 7. A associação entre atores '*INS*' só deve ocorrer entre Agentes para representar uma instanciação[4].

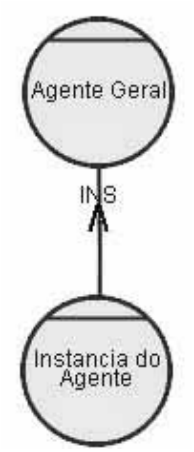


Figura 46 - Guideline 7: associação INS somente entre Agentes para representar instanciação.

Guideline 8. Um *link* de Associação só deve ser criado entre atores. Não pode haver, por exemplo, um link de associação entre um ator e um elemento de dependência, como ilustrado na *Figura 47*.

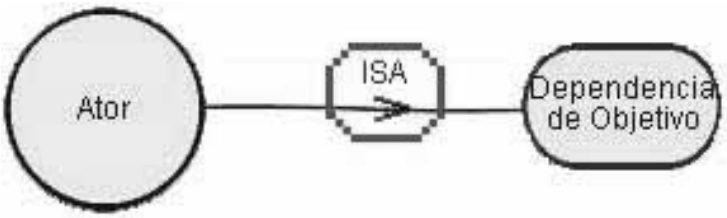


Figura 47 - Guideline 8: Link de associação somente entre atores.

[4] Instanciação é usada para representar o relacionamento entre entidades mais específicas com uma entidade mais geral. Em i* é utilizada para associar Agentes. Por exemplo: no modelo organizacional da conferência pode-se criar os agentes Revisor e *Chair* e associá-los a um agente mais geral (Membro do Comitê).

Guideline 9. Não deve haver dependência cíclica no modelo. Pois desta forma não fica claro como a dependência será satisfeita.

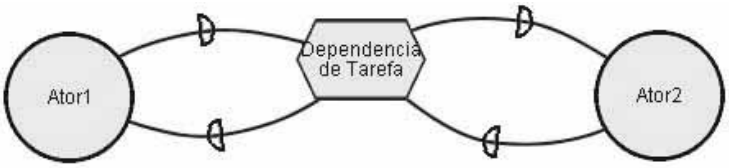


Figura 48 - Guideline 9: Não deve haver dependência cíclica.

Guideline 10. Os *links* de dependência que representam uma relação entre atores devem seguir a mesma direção.

Figura 49 - Guideline 10: Links de dependência devem seguir a mesma direção.

Guideline 11. Não usar *link* de dependência entre atores sem mostrar o *Dependum*.

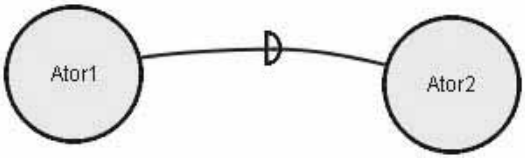


Figura 50 - Guideline 11: não usar links de dependência entre atores sem mostrar o Dependum.

Guideline 12. Não se deve reusar um *Dependum* em mais de uma relação de dependência.

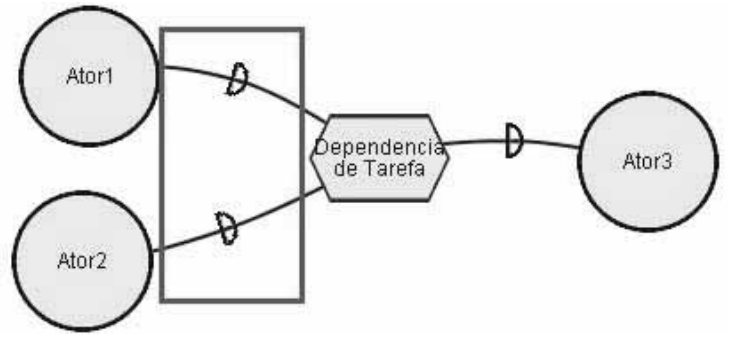


Figura 51 - Guideline 12: não se deve usar o Dependum e mais de uma relação de dependência.

Guideline 13. Pode existir uma relação de dependência do tipo *One-Side* - unilateral. Ou seja, é uma relação que pode ser estabelecida de um *Depender* para um *Dependum* ou de um *Dependum* para um *Dependee*.

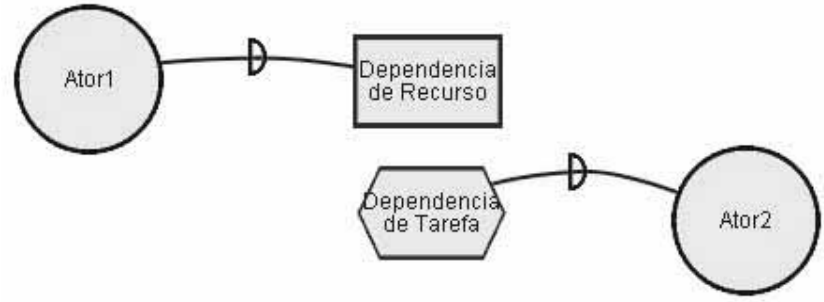


Figura 52 - Guideline 13: pode haver dependência do tipo One-side.

Guideline 14. Deve-se usar o símbolo “D” para denotar um *link* de Dependência.

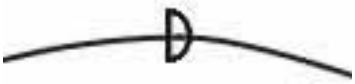


Figura 53 - Guideline 14: símbolo do link de dependência.

Modelo SR

Como já apresentado o modelo SR é um diagrama com nós e *links* que juntos fornecem uma estrutura que expressa as razões das dependências entre atores. Os atores modelados em um modelo SD são refinados a fim de que suas intenções específicas sejam visualizadas. Existem quatro tipos de nós, semelhante ao modelo SD, utilizados para representar dependências: objetivo, tarefa, recurso e objetivo-*soft*. E são adicionadas três classes de ligação internos ao ator i*: ligação meio-fim, decomposição de tarefa e *links* de contribuição.

Guideline 15. Um ator não pode existir dentro da fronteira de um outro ator.

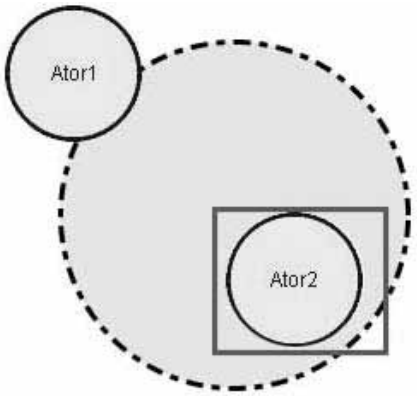


Figura 54 - Guideline 15: não deve existir um ator na fronteira de outro ator.

Guideline 16. Não se deve usar *links* de dependência dentro da fronteira de um ator.

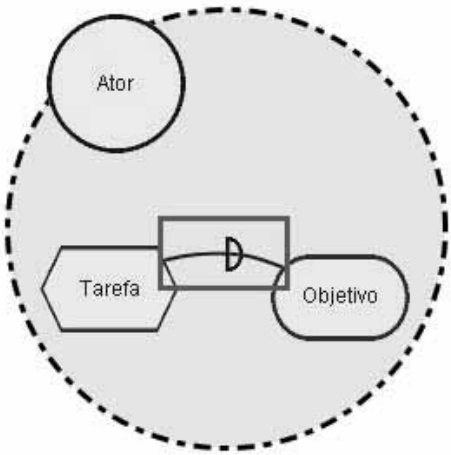


Figura 55 - Guideline 16: não se deve usar links de dependência dentro da fronteira de um ator.

Guideline 17. Um objetivo só pode ser decomposto usando a ligação meio-fim. Ou seja, para indicar que um Objetivo pode ser atingido através da execução de diferentes Tarefas, deve-se modelar a relação usando a ligação de meio-fim. O uso dessa ligação indica alternativa, e só deve ser usado para ligar Tarefas a Objetivos.

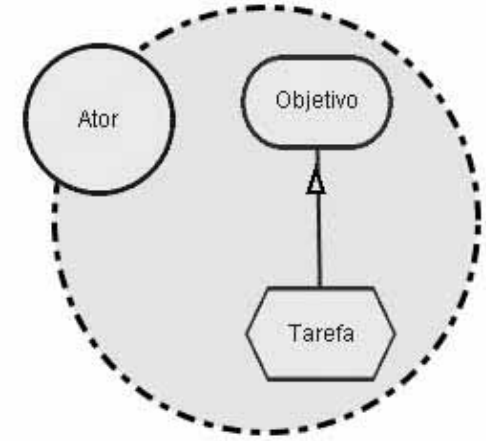


Figura 56 - Guideline 17: ligação meio-fim usado para ligar somente tarefas a objetivos.

Guideline 18. Uma tarefa só pode ser decomposta usando a ligação de decomposição de tarefas. Essa ligação só deve ser usada para ligar uma Tarefa a outras Tarefas, Objetivos, Objetivo-*Soft* ou Recurso.

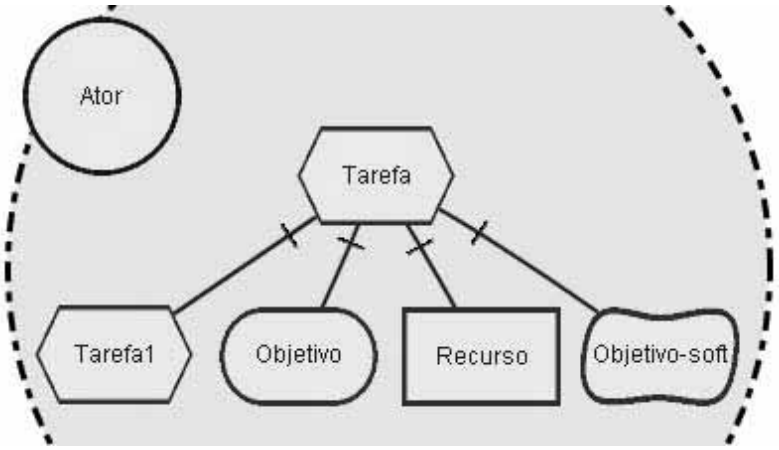


Figura 57 - Guideline 18: link de decomposição de tarefa.

Guideline 19. As ligações de decomposição e meio-fim só devem ser utilizados dentro da fronteira do ator.

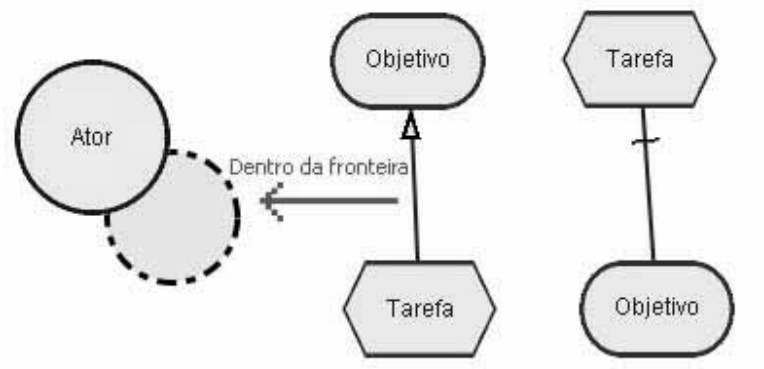


Figura 58 - Guideline 19: ligação meio-fim e decomposição de tarefa apenas na fronteira do ator.

Guideline 20. Objetivos-*soft* só podem ser refinados usando-se *links* de contribuição. O *link* é usado de qualquer elemento para um Objetivo-soft.

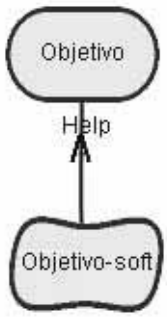


Figura 59 - Guideline 20: usar links de contribuição para relacionar qualquer elemento a um objetivo-soft.

4.3.2 Definição do Modelo de Domínio

Como apresentado na seção anterior, a definição de um meta-modelo é o primeiro passo para a construção de um editor gráfico. O meta-modelo segundo o processo do *framework* GMF, é definido no formato de um arquivo XMI denominado Ecore. Este arquivo representa o modelo do domínio, ou seja, descreve o que é e como se comporta um modelo em i*. Para o i* [i* Wiki 2008][Grau 2008] foi criado o modelo apresentado na *Figura 60*. Este modelo, como já mencionado na seção anterior, também pode ser visualizado graficamente como um diagrama de classes. O diagrama para o modelo Ecore criado pode ser visualizado na *Figura 61*.

Como já mencionado na seção 3.1.1, o i* está sendo utilizado por vários grupos de pesquisa o que ocasionou no surgimento de diversas variantes da linguagem. Como nossa dissertação toma como referência o i* segundo [i* Wiki 2008][Grau 2008], apresentaremos no Apêndice A uma comparação entre o meta-modelo criado nesta dissertação com alguns trabalhos publicados que referenciam as variações do i* [Ayala 2006][Lucena 2008].

O modelo apresentado nas figuras 60 e 61 - criado para essa dissertação, indica que cada modelo i* criado é composto de um conjunto de nós e um conjunto de relacionamentos entre esses nós. Cada nó pode representar dois conceitos diferentes: um Ator ou um Elemento. Um Ator no modelo pode ser: Ator, Agente, Papel, ou Posição. E um elemento no modelo pode ser: um Objetivo, uma Tarefa, um Recurso, ou um Objetivo-Soft.

Cada relacionamento pode ser:

(i) Um *link* de Dependência: relacionamento entre Atores e Elementos. Podendo esses *links* de dependência ter graus de vulnerabilidade, ou de força: aberto, fechado, ou crítico.

(ii) Uma Associação entre atores: relacionamento somente entre os diferentes tipos de atores.

(iii) Um *link* de relação Meio-Fim: relacionamento entre tarefas e objetivos.

(iv) Um *link* de Decomposição de Tarefas: relacionamento entre os diferentes elementos (tarefas, recursos, objetivos, objetivos-*soft*) a tarefas.

(v) Um *link* de contribuição: relacionamento entre objetivos, tarefas e objetivos-*soft,* com objetivos-*soft*.

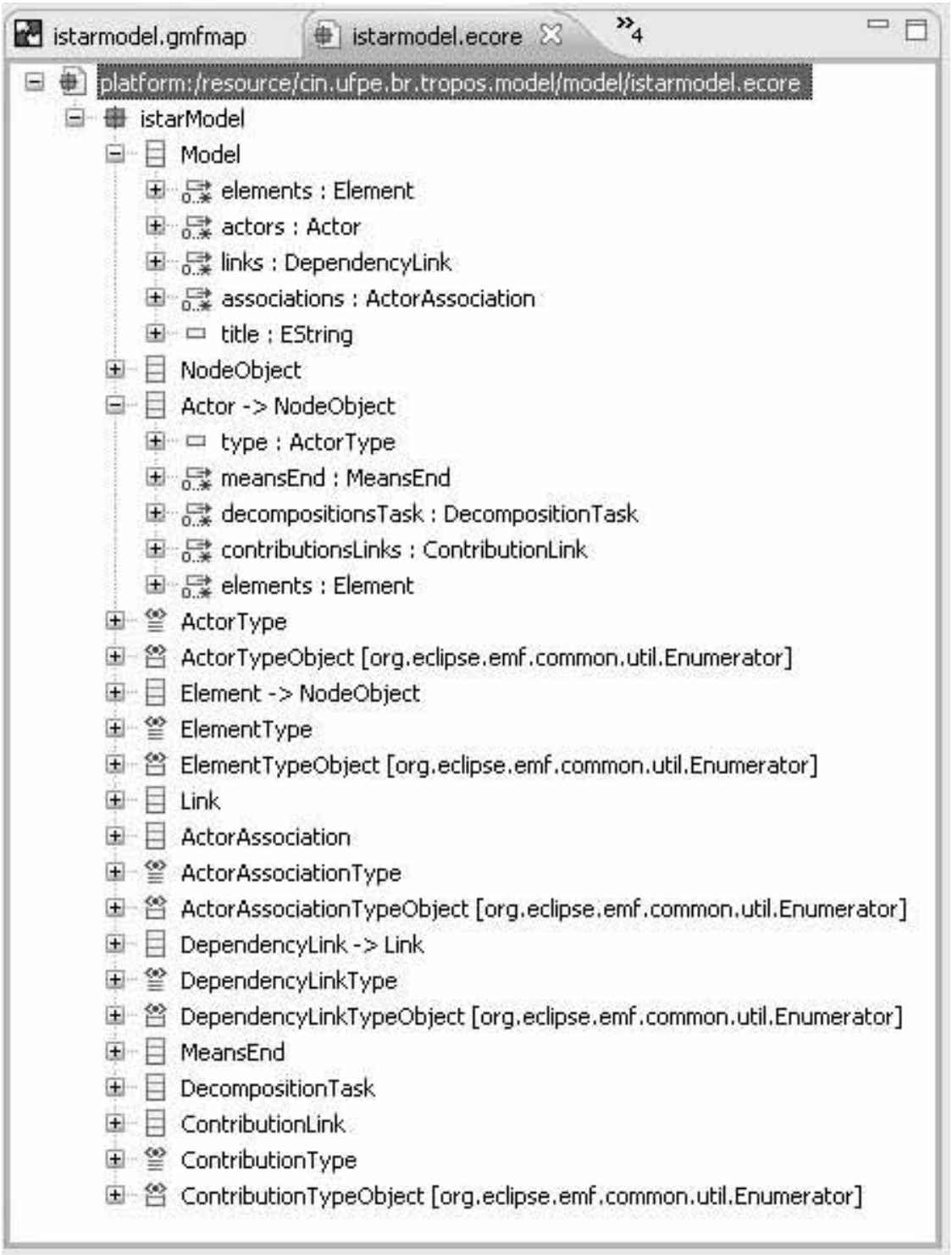


Figura 60 - Modelo ecore para o i* segundo [i* Wiki 2008][Grau 2008].

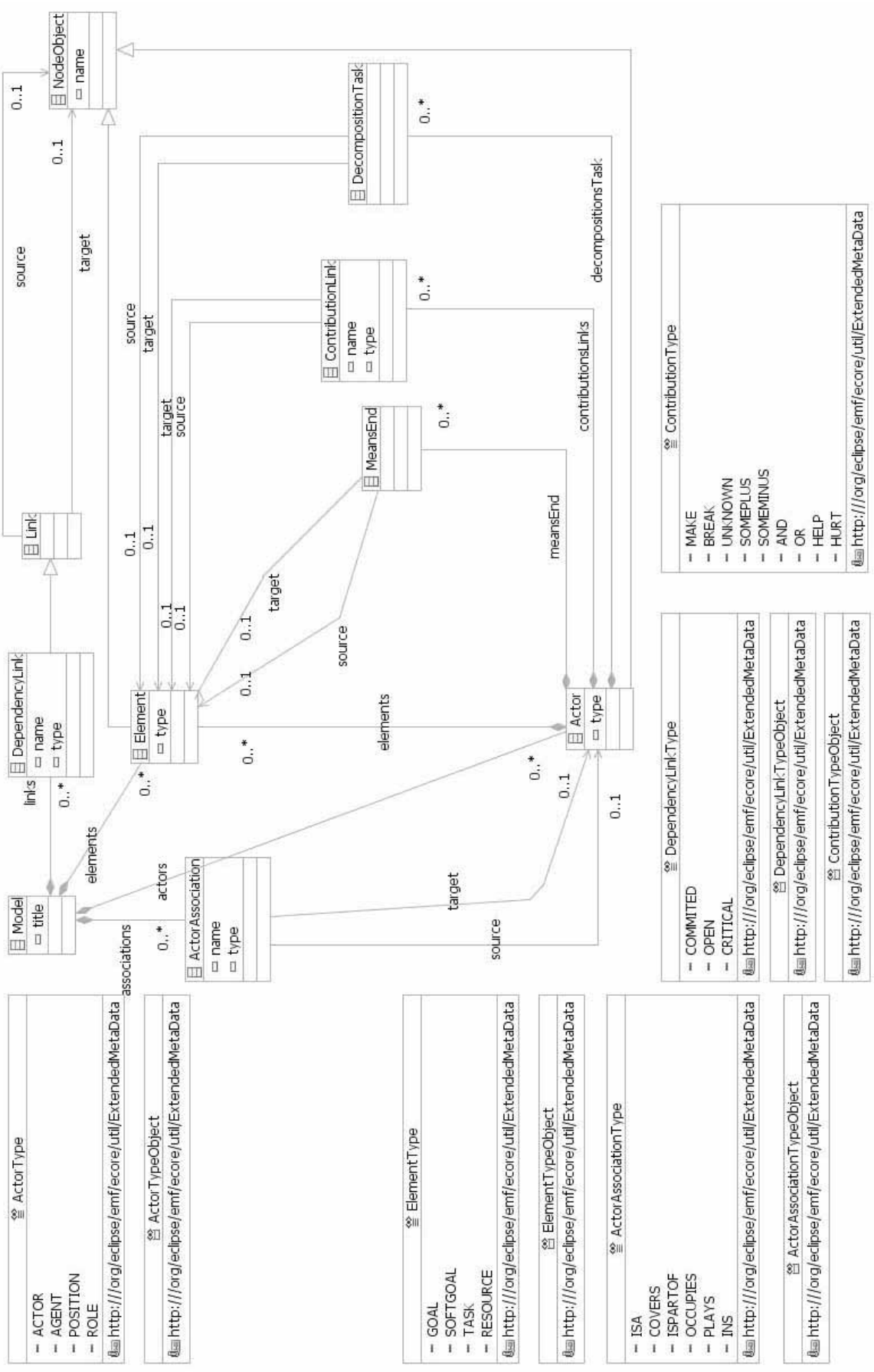


Figura 61 - Modelo de classes para o i* segundo [i* Wiki 2008][Grau 2008].

Uma vez que o meta-modelo foi construído, o modelo EMF GenModel foi criado e dele foi gerado o código que implementa o meta-modelo do domínio - através da opção *Generate Model Code* e *Generate Edit Code*, *Figura 62*. Esse código nada mais é do que as classes, em linguagem Java, que representam os elementos de domínio definidos. A opção *Generate Model Code* gera as classes básicas para cada elemento dentro da pasta *src* no mesmo projeto onde o meta-modelo foi criado - *cin.ufpe.br.tropos.model*. E a opção *Generate Edit Code* gera as classes provedoras de acesso às propriedades de cada elemento dentro do projeto *cin.ufpe.br.tropos.model.edit*.

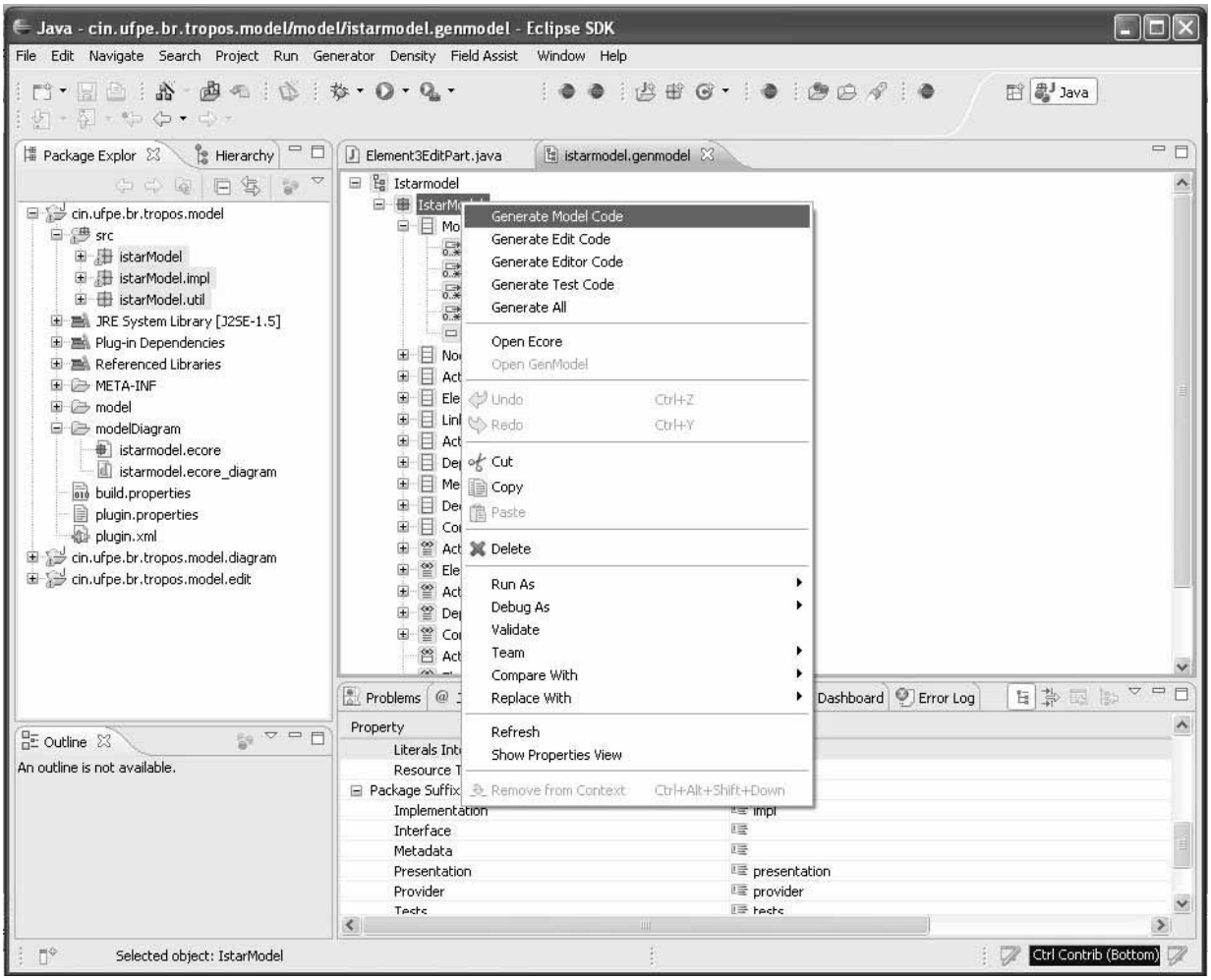


Figura 62 - Modelo EMF GenModel: geração de código do modelo de domínio.

Seguindo o processo do *framework* GMF (vide seção 4.2 e a *Figura 31*), após a geração do código, definimos os modelos: gráfico, ferramental, de mapeamento e de geração - que serão detalhados nas seções abaixo.

4.3.3 Definição do Modelo Gráfico

O modelo gráfico, como citado na sessão anterior, tem o propósito de definir os elementos gráficos de um diagrama. Na *Figura 63* apresentamos o modelo GMFGraph, com a definição das figuras, dos nós, *labels* (rótulos), e *links* de relacionamento da ferramenta que está sendo criada.

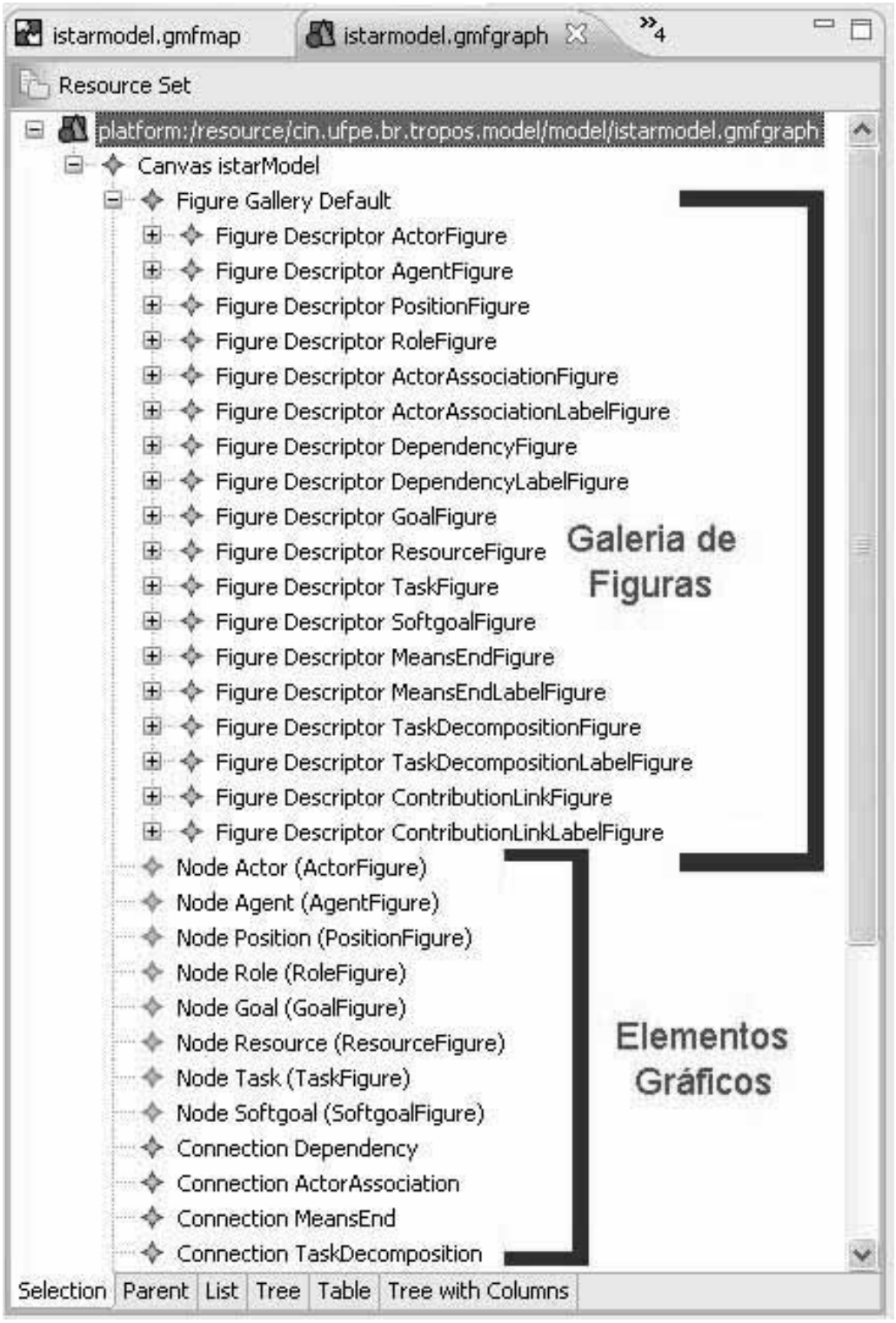


Figura 63 - Modelo gráfico da ferramenta de modelagem i*.

Na *Figura 63* podemos visualizar:

(i) A galeria de figuras (*Figure Gallery Default*) onde se encontram todas as figuras (*Figure Descriptor*) que definem as formas dos elementos e que serão utilizadas pelos elementos do diagrama:

a. Figuras dos atores: Ator (*ActorFigure*), Agente (*AgentFigure*), Posição (*PositionFigure*), e Papel (*RoleFigure*).

b. Figuras dos elementos de dependência: Objetivo (*GoalFigure*), Recurso (*ResourceFigure*), Tarefa (*TaskFigure*), e Objetivo-soft (*SoftgoalFigure*).

c. Figura do link de dependência: link de dependência (*DependencyFigure*), rótulo que indica vulnerabilidade (*DependencyLabelFigure*).

d. Figura do link de associação: link de associação (*ActorAssociationFigure*), rótulo que indica o tipo de associação (*ActorAssociationLabelFigure*).

e. Figura do link de ligação meio-fim: *MeansEndFigure*.

f. Figura do link de decomposição de tarefas: *TaskDecompositionFigure*.

g. Figura do link de contribuição: *ContributionLinkFigure*.

(ii) Os elementos que definem os nós (*Nodes*), os rótulos (*Diagram Label*), e os relacionamentos (*Connection*).

A *Figura 64* ilustra em detalhes a descrição gráfica do elemento de dependência Recurso na galeria de figuras. Para este elemento foi criado um retângulo que tem como propriedades o tamanho - *Size: (60,50)*, a cor da borda - *Foreground: {0,0,0}*, e a cor de fundo - *Background: {204,255,204}*. Outra definição realizada foi a criação de um label para a figura (*Label ResourceNameFigure*), e a criação de acesso ao label da figura (*Child Access getFigureResourceNameFigure*).

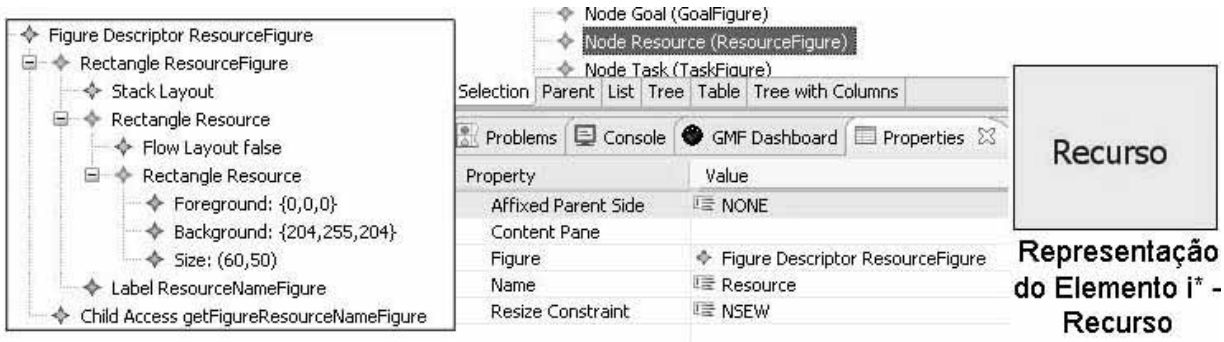


Figura 64 - Descrição da figura (Figure Descriptor) do elemento de dependência Recurso.

Como apresentado o *framework* GMF fornece um *wizard* para a criação de Modelos Gráficos. Isto permite a criação de nós, *labels* e conexões padrões a partir de um conjunto de padrões que são fornecidos pelo assistente. O assistente nos retorna um Modelo Gráfico bem simples, mas que pode ser modificado a fim de produzir o comportamento desejado. Portanto, para a nossa ferramenta foi preciso modificar esse modelo a fim de que este refletisse o ambiente esperado.

Em particular, foi preciso definir novas figuras na Galeria de Figuras e realizar a configuração apropriada dos nós para as novas figuras. Para a criação das novas figuras foram usados retângulos, elipses e polígonos. Os retângulos foram usados na definição de todos os nós i* do modelo; as elipses foram usadas na configuração dos atores, objetivos e objetivos-*soft*; e os polígonos foram usados na criação dos nós que representam as tarefas.

Outra modificação necessária foi a aplicação de estilos diferentes nas conexões. Foram realizadas essencialmente alterações no estilo das linhas e no estilo das pontas das linhas de conexão. Para a linha foram alteradas a largura, a cor, a utilização de rótulo (para representar as associações entre atores e *links* de contribuição), e os elementos gráficos a serem utilizados nas extremidades. Para as pontas da linha (ou *Decoration*, como é utilizado no *framework* GMF) foi utilizada a Decoração Customizada, através da qual é criada uma classe que implementa a figura que se deseja como extremidade de uma linha.

Para a ligação de dependência, por exemplo, foi criada uma classe que implementa o elemento gráfico que indica o sentido da dependência (vide *Figura 65*). A classe criada, *DependencyDecoration*, implementa o método *paintFigure(Graphics)* da interface *RotatableDecoration*, esta interface é a mesma utilizada pelo GMF para gerar o elemento padrão (*Polyline Decoration*) para decoração das extremidades de uma conexão. O principal método dessa classe, que pode ser visto na figura, é o *paintFigure(Graphics)*, é este o método que desenha a figura do link de dependência.

DependencyDecoration.java

```
package istarModel.diagram.edit.parts;

import org.eclipse.draw2d.Graphics;

public class DependencyDecoration extends RectangleFigure implements RotatableDecoration {

    private Point location = new Point();
    private Transform transform = new Transform();
    private static final Dimension SIZE = new Dimension(15, 17);

    public void paintFigure(Graphics g) {
        Rectangle r = getBounds().getCopy();
        r.translate(2, 2);
        r.setSize(11, 9);

        //draw main area
        g.fillRectangle(r);

        //outline main area
        g.drawLine(r.x, r.y, r.right() - 1, r.y);
        g.drawLine(r.right() - 1, r.y, r.right() - 1, r.bottom() - 1);
        g.drawLine(r.x, r.y, r.x, r.bottom() - 1);

        //draw and outline the arc 'D'
        r.height = 9;
        r.y += 4;
        g.fillArc(r, 180, 180);
        r.width--;
        r.height--;
        g.drawArc(r, 180, 190);

    }

    public void setLineWidth(int width){
```

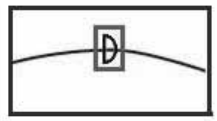


Figura 65 - Implementação customizada de decoração para o link de dependência.

4.3.4 Definição do Modelo Ferramental

Após a definição do modelo gráfico, criamos o Modelo Ferramental. Como discutido anteriormente na seção 4.2.3, este modelo é utilizado para definir o *toolbar* e os menus a serem usados na ferramenta que está sendo criada. Para nossa ferramenta definimos o *toolbar*, que é o menu default criado pelo *framework* GMF. Nele, foram definidos dentro da paleta (*Palette*) os grupos de ferramentas (*Tool Groups*), contendo a definição das opções (*Tools*) a serem exibidas na paleta - como ilustrado na *Figura 66* e detalhado abaixo.

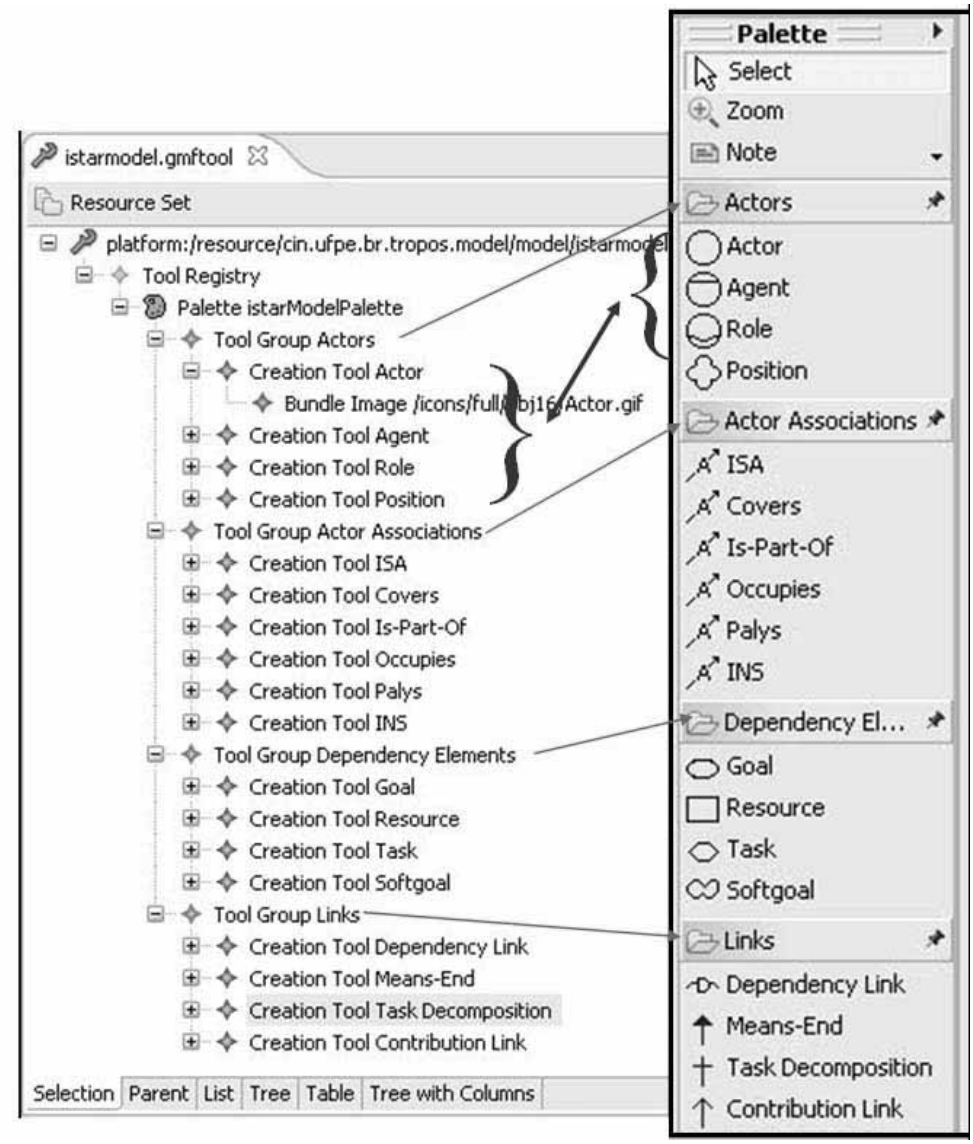


Figura 66 - Definição do Modelo Ferramental.

Para a definição do modelo ferramental o GMF também fornece um assistente para gerar o modelo automaticamente. Como já apresentado, os modelos gerados automaticamente produzem um conjunto padrão que pode ser modificado a fim de que este modelo forneça o comportamento desejado. A alteração realizada (vide *Figura 66*) em cima do modelo gerado para a ferramenta foi a criação de grupos de ferramentas (*Tool Groups*) para agrupar os elementos. Os grupos de ferramentas criados foram: Atores (*Actors*), Associações de Atores (Actor *Associations*), Elementos de Dependência (*Dependency Elements*), e *Links*.

Para cada grupo as seguintes opções de ferramentas (*Creation Tool*) foram criadas (vide *Figura 66*):

(i) Grupo Atores (*Tool Group Actor*): Ator (*Actor*), Agente (*Agent*), Papel (*Role*), Posição (*Position*);

(ii) Grupo de Associação entre atores (*Tool Group Actor Associations*): ISA, Covers, Is-Part-Of, Occupies, Plays, INS;

(iii) Grupo de Elementos de Dependência (*Tool Group Dependency Elements*): Objetivo (*Goal*), Recurso (*Resource*), Tarefa (*Task*), Objetivo-*soft* (*Softgoal*);

(iv) Grupo de Links (*Tool Group Links*): *Link* de Dependência (*Dependency Link*), Ligação Meio-Fim (*Means-End*), Decomposição de Tarefa (*Decomposition Task*), *Link* de Contribuição (*Contribution Link*).

No modelo gerado pelo *framework* GMF é criada uma imagem default a ser utilizada em todos os elementos criados para exibição na paleta. Portanto, para cada opção da paleta foram elaboradas imagens a serem associadas às respectivas opções. As imagens criadas são associadas através da opção *Bundle Image*, como mostrado na *Figura 66*, bastando apenas configurar o caminho correto para que cada imagem criada seja visualizada na paleta quando a ferramenta for executada.

4.3.5 Criação do Modelo de Mapeamento

O Modelo de Mapeamento, como já apresentado na seção 4.2.4, é usado no GMF para realizar o mapeamento entre o que foi definido nos modelos gráfico e ferramental com o que foi definido no meta-modelo. Este modelo pode ser automaticamente gerado através de um assistente do GMF. Contudo, como já mencionado nesta seção na Definição do Modelo Gráfico e Definição do Modelo Ferramental, a geração automática dos modelos através do GMF é bastante simples e no caso do modelo de mapeamento é pouco confiável, pois este pode gerar um mapeamento que reflita em um comportamento não esperado e, portanto este modelo deve ser verificado.

No modelo de mapeamento, como pode ser visualizado na *Figura 67*, encontram-se as definições dos nós (*Top Node Reference*), das conexões (*Link Mapping*) e das restrições (*Audit Container*) que serão aplicadas.

Os nós (chamados *Top Node Reference* - vide *Figura 67*) são usados para mapear os nós no diagrama a: elementos no meta-modelo, figuras definidas no modelo gráfico, e ferramentas de criação dos nós. Para cada nó é adicionado um *Node Mapping*, onde são definidos o mapeamento dos rótulos (*Label Mappings*), das referências a elementos filhos (*Child References*), e dos compartimentos (*Compartment Mappings*).

Os links (chamados *Link Mapping* - vide *Figura 67*) são usados para mapear as conexões no diagrama a: atributos dos elementos no meta-modelo, figuras definidas no modelo gráfico, e ferramentas de criação dos link.

As retrições (chamadas *Audit Rule* - vide *Figura 67*) foram definidas no mapeamento dentro de um *Audit Container*. Este elemento possibilita a criação de várias restrições que podem ser acionadas para

validação na criação dos modelos na ferramenta que está sendo criada - esse acionamento pode ser automático (à medida que os elementos são criados) ou manual (acionando a opção *Diagram > Validate* no menu).

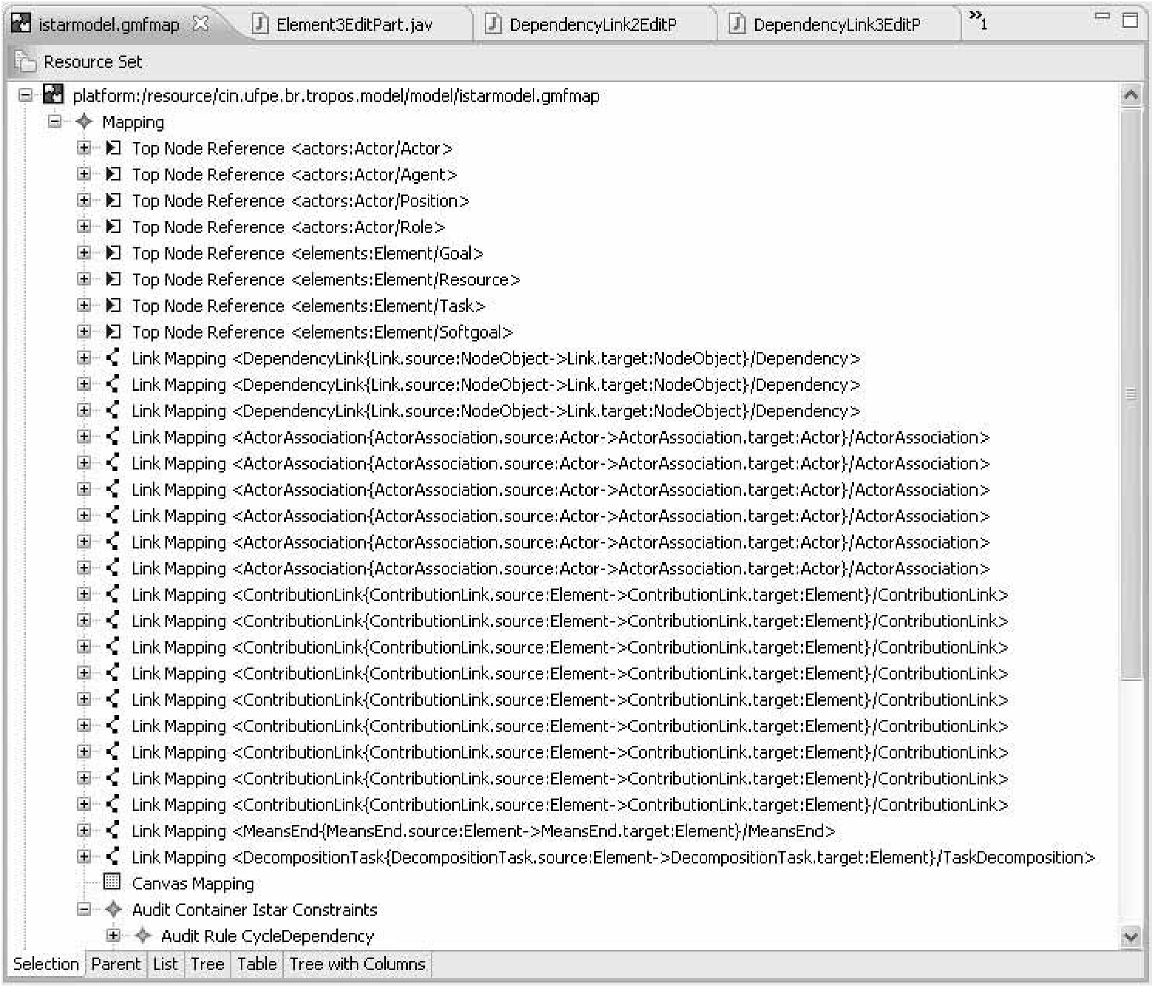


Figura 67 - Modelo de mapeamento da ferramenta de modelagem i*.

A seguir damos exemplo da definição de um nó, de um link e de uma restrição no modelo de mapeamento:

(i) Mapeamento do nó referente ao elemento de dependência Recurso (vide *Figura 68*).

No mapeamento do nó Recurso as seguintes propriedades foram definidas: (a) *Domain meta information - Element* que associa o nó ao elemento no meta-modelo, (b) *Visual representation - Diagram Node* que associa o nó à figura definida no Modelo Gráfico, (c) *Visual representation - Tool* que mapeia o nó à ferramenta definida no Modelo Ferramental. Além dessas propriedades foram adicionadas ao nó Recurso: (a) uma restrição que indica qual o tipo do elemento quando ele for criado (*ElementType::RESOURCE*) e o valor inicial do label no nó (*Feature Seq Initializer - Feature Value Spec<name:='resource'>*), e (b) um mapeamento para o atributo que indica o label do elemento (*Feature Label Mapping*).

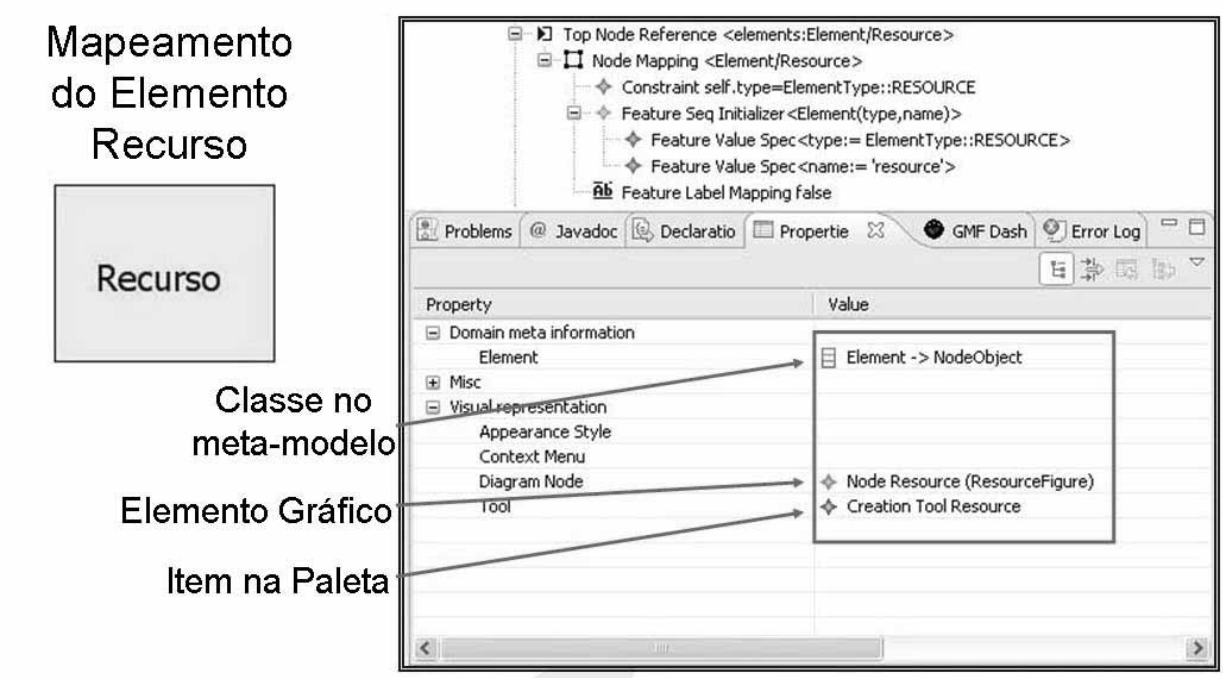


Figura 68 - Exemplo de Mapeamento de um Nó no Modelo de Mapeamento.

(ii) Mapeamento do link referente a uma Associação entre atores (vide *Figura 69*).

No mapeamento do link de Associação entre atores *ISA* as seguintes propriedades foram definidas: (a) *Domain meta information - Containment Feature* que associa o link ao atributo *associations* do elemento *Model* definido no meta-modelo, (b) *Visual representation - Diagram Link* que associa o link à figura definida no Modelo Gráfico, (c) *Visual representation - Tool* que mapeia o link à ferramenta definida no Modelo Ferramental. Além dessas propriedades foram adicionadas ao link de associação *ISA*: (a) uma restrição que indica qual o tipo da associação quando ela for criada (*ActorAssociationType::ISA*) e o valor inicial do label no link (*Feature Seq Initializer - Feature Value Spec<name:='ISA'>*), (b) um mapeamento para o atributo que indica o label do elemento (*Feature Label Mapping*), e (c) uma restrição à criação do link que o impede de ligar um elemento a ele mesmo (*Link Constraint - Constraint self <> oppositeEnd*).

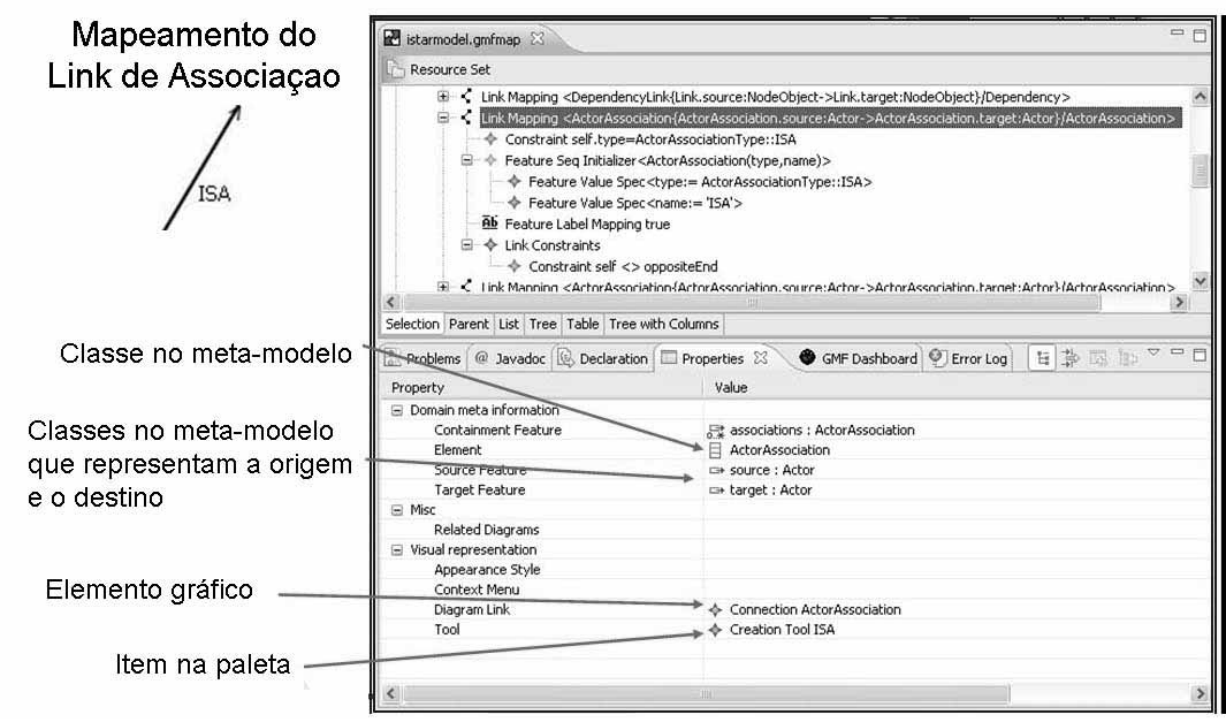


Figura 69 - Exemplo de Mapeamento de um Link no Modelo de Mapeamento.

(iii) Mapeamento da restrição refente ao uso de link de Dependência apenas entre atores e elementos de dependência, não se devendo usar *link* de dependência entre atores sem mostrar o *Dependum* (vide *Figura 70*) - Guideline 11.

No mapeamento de restições são criadas várias regras, chamadas de *Audit Rule*. Para cada regra são definidos o elemento do meta-modelo que será o alvo da restrição (*Domain Element Target*) e o corpo da restrição (*Constraint*) segundo a linguagem escolhida. Para a restrição do uso do link de dependência o elemento alvo da restrição é o elemento *Model* definido no meta-modelo. Quanto ao corpo da restrição, a linguagem utilizada para sua descrição foi OCL. A regra portanto ficou assim descrita:

self.links -> forAll(link1 | (link1.target.oclIsTypeOf(Actor) implies link1.source.oclIsTypeOf(Element)) and (link1.target.oclIsTypeOf(Element) implies link1.source.oclIsTypeOf(Actor)))

As demais restrições em OCL criadas no modelo de mapeamento, na definição do elemento *Audit Container*, são apresentadas a seguir.

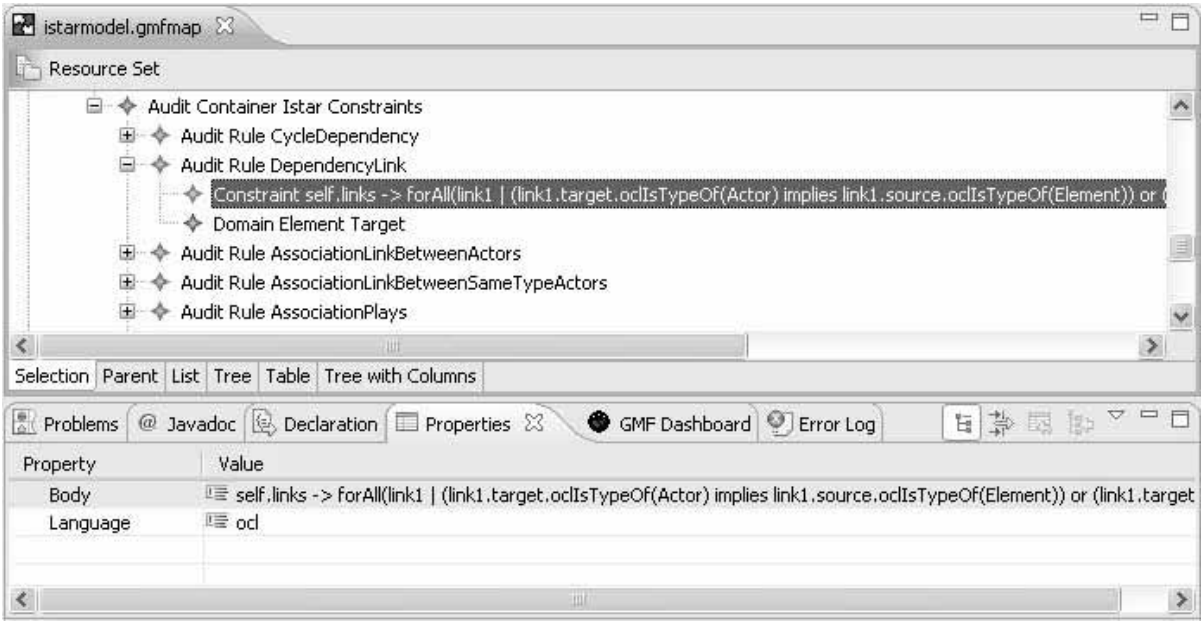


Figura 70 - Exemplo de Mapeamento de uma restrição no Modelo de Mapeamento.

Restrições Definidas em OCL

No modelo de mapeamento foi criado um conjunto de restrições que tiveram como base o guia (contendo *guidelines* e boas práticas) [Grau 2008] fornecido em [i* Wiki 2008], e alguns itens do catálogo de erros mais comuns construído em [Santos 2008]. Não contemplamos todos os itens do catálogo de [Santos 2008] pois alguns deles dependem da interpretação do conteúdo dos elementos, por exemplo: o uso adequado de palavras para nomear os elementos do modelo (vide Anexo A). Nossa preocupação é focar em um conjunto de restrições básicas sobre o uso correto das ligações dos elementos na construção de modelos em i*.

Algumas das regras foram contempladas na definição do meta-modelo em Ecore criado para o i* segundo [i* Wiki 2008][Grau 2008] e as demais foram definidas no modelo de mapeamento utilizando OCL como linguagem de descrição. Como referência para criação das regras em OCL foram utilizados o tutorial disponível em [Tutorial OCL 2005] e a especificação da linguagem disponível no site da OMG [OCL 2007].

Como regras definidas no modelo de mapeamento, temos:

(i) A principal restrição criada para o mapeamento das ligações é a restrição relativa a um elemento (em OCL, *self*) não poder se ligar a ele mesmo: o nó fim da ligação (*oppositeEnd*) não pode ser ele mesmo (*self*). Seja este elemento um ator ou um elemento de dependência (vide *Figura 71*).

Tabela 7 - Regra OCL: um elemento não pode se ligar a ele mesmo.

```
self <> oppositeEnd
```

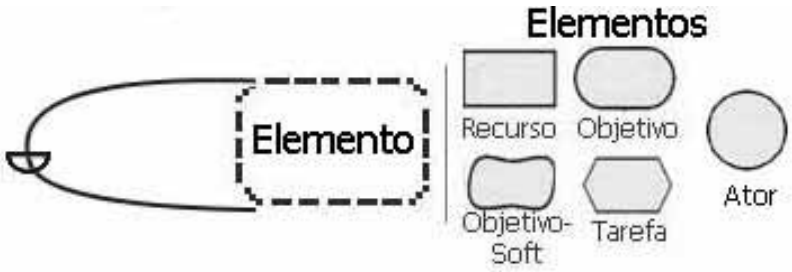


Figura 71 - Regra OCL: um elemento não pode se ligar a ele mesmo.

(ii) Outra restrição importante é a não possibilidade de criação de uma dependência cíclica.

Tabela 8 - Regra OCL: impossibilidade de criação de dependência cíclica.

```
self.links -> forAll(link1, link2 | link1.target = link2.source
and link1.type = link2.type implies link2.target <> link1.source)
```

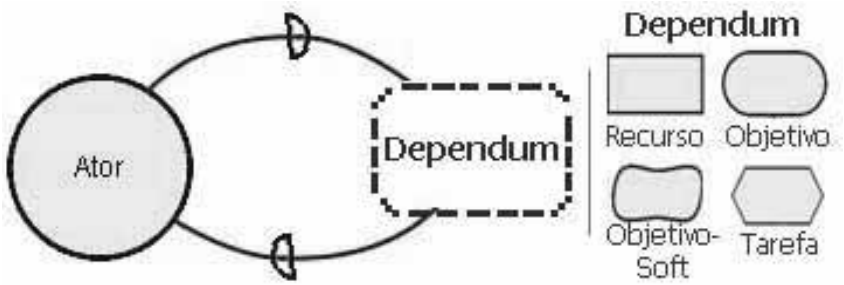


Figura 72 - Regra OCL: impossibilidade de criação de dependência cíclica.

(iii) Para o *link* de dependência só é permitida a ligação entre atores e elementos de dependência no caso de modelo SD. Não usar *link* de dependência entre atores sem mostrar o *Dependum*.

Tabela 9- Regra OCL: não usar link de dependência entre atores sem mostrar o Dependum.

```
self.links -> forAll( link1 | ( link1.target.oclIsTypeOf(Actor)
implies link1.source.oclIsTypeOf(Element)) and
(link1.target.oclIsTypeOf(Element) implies
link1.source.oclIsTypeOf(Actor)) )
```

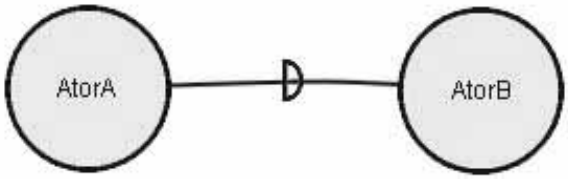


Figura 73 - Regra OCL: não usar link de dependência entre atores sem mostrar o Dependum.

(iv) A utilização do *link* de associação só é permitida para nós do tipo ator.

Tabela 10 - Regra OCL: uso de link de associação apenas entre elementos do tipo ator.

```
self.associations -> forAll( association |
association.source.oclIsTypeOf(Actor) implies
association.target.oclIsTypeOf(Actor))
```

Figura 74 - Regra OCL: uso de link de associação apenas entre elementos do tipo ator.

(v) A associação entre atores do tipo 'IS A' e 'Is-Part-Of' só pode ser realizada entre atores do mesmo tipo.

Tabela 11 - Regra OCL: associação 'IS A' e 'Is-Part-Of' apenas entre atores do mesmo tipo.

```
self.associations -> forAll(association |
(association.type=ActorAssociationType::ISA or
association.type=ActorAssociationType::ISPARTOF) implies
(association.source.type = association.target.type) )
```

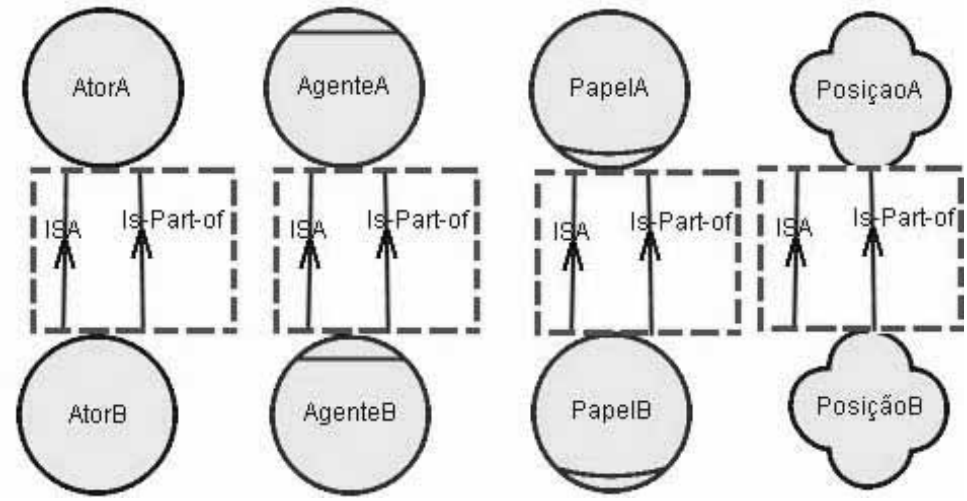


Figura 75 - Regra OCL: associação 'IS A' e 'Is-Part-Of' apenas entre atores do mesmo tipo.

(vi) A associação entre atores '*Plays*' só deve ocorrer de um Agente para um Papel.

Tabela 12 - Regra OCL: associação 'Plays' apenas de um Agente para um Papel.

```
self.associations -> forAll(association |
association.type=ActorAssociationType::PLAYS implies
association.source.type=ActorType::AGENT and
association.target.type=ActorType::ROLE )
```

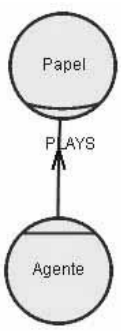


Figura 76 - OCL: associação 'Plays' apenas de um Agente para um Papel.

(vii) A associação entre atores '*Covers*' só deve ocorrer de uma Posição para um Papel.

Tabela 13 - Regra OCL: associação 'Covers' apenas de uma Posição para um Papel.

```
self.associations -> forAll(association |
association.type=ActorAssociationType::COVERS implies
association.source.type=ActorType::POSITION and
association.target.type=ActorType::ROLE )
```

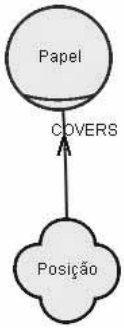


Figura 77 - Regra OCL: associação 'Covers' apenas de uma Posição para um Papel.

(viii) A associação entre atores '*Occupies*' só deve ocorrer de um Agente para uma Posição.

Tabela 14 - Regra OCL: associação 'Occupies' apenas de um Agente e uma Posição.

```
self.associations -> forAll(association |
association.type=ActorAssociationType::OCCUPIES implies
association.source.type=ActorType::AGENT and
association.target.type=ActorType::POSITION )
```

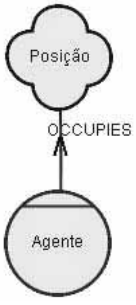


Figura 78 - Regra OCL: associação 'Occupies' apenas de um Agente e uma Posição.

(ix) Não se deve reusar um *Dependum* em mais de uma relação de dependência.

Tabela 15 - Regra OCL: não reusar um Dependum em mais de uma relação.

```
self.links -> forAll(link1 | not self.links -> exists(link2 |
link1 <> link2 and ( (link1.source = link2.source and
link1.source.oclIsTypeOf(Element)) or(link1.target = link2.target
and link1.source.oclIsTypeOf(Element)) )  ) )
```

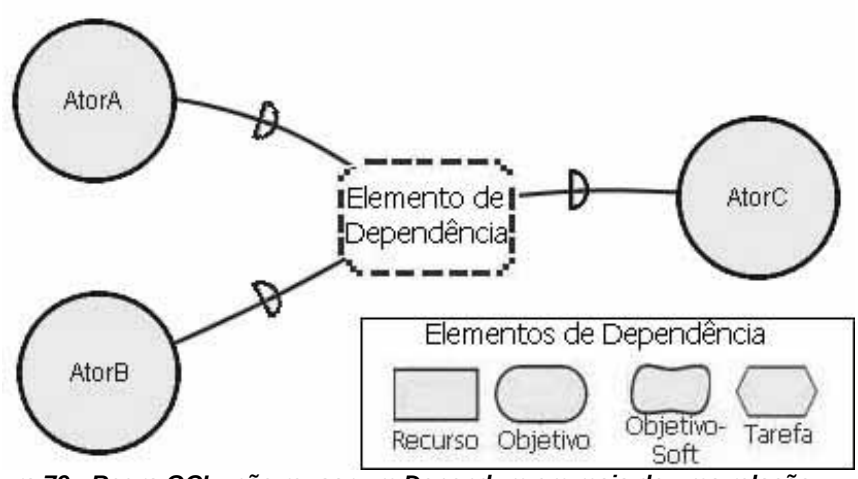


Figura 79 - Regra OCL: não reusar um Dependum em mais de uma relação.

(x) Um objetivo só pode ser decomposto usando a ligação meio-fim. Ou seja, para indicar que um Objetivo pode ser atingido através da execução de diferentes Tarefas, deve-se modelar a relação usando a ligação de meio-fim. O uso dessa ligação só deve ser usado para ligar Tarefas a Objetivos.

Tabela 16 - Regra OCL: uso de ligação meio-fim.

```
self.meansEnd -> forAll(link | link.source.type=ElementType::TASK
and link.target.type=ElementType::GOAL)
```

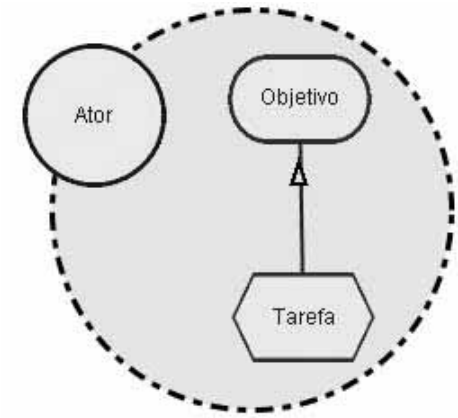


Figura 80 - Regra OCL: uso de ligação meio-fim.

(xi) Uma tarefa só pode ser decomposta usando o *link* de decomposição de tarefas. Esse *link* só deve ser usado para ligar uma Tarefa a outras Tarefas, Objetivos, Objetivo-*Soft* ou Recurso.

Tabela 17 - Regra OCL: uso de link de decomposição de tarefas.

```
self.decompositionsTask -> forAll(decomposition |
decomposition.target.type=ElementType::TASK and
decomposition.source.oclIsTypeOf(Element))
```

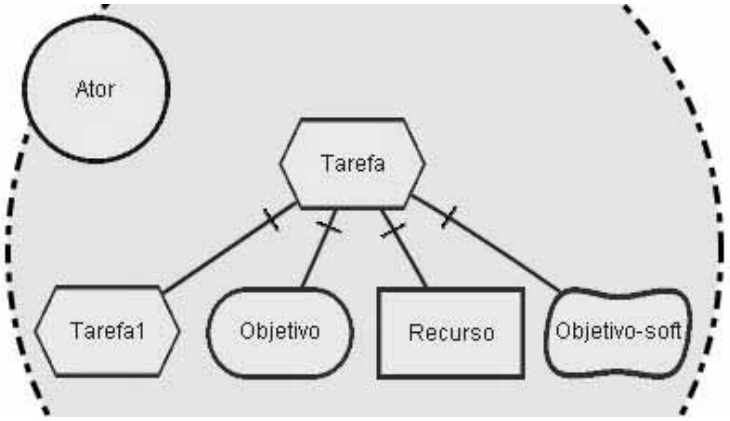


Figura 81 - Regra OCL: uso de link de decomposição de tarefas.

(xii) Não deve haver nós (atores ou elementos de dependência) sem ligação no modelo.

Para o nó que corresponde a um elemento de dependência no modelo:

Tabela 18 - Regra OCL: não deve haver elementos de dependência sem ligação no modelo.

```
self.elements -> forAll(element | self.links -> exists(link |
link.source=element or link.target=element))
```

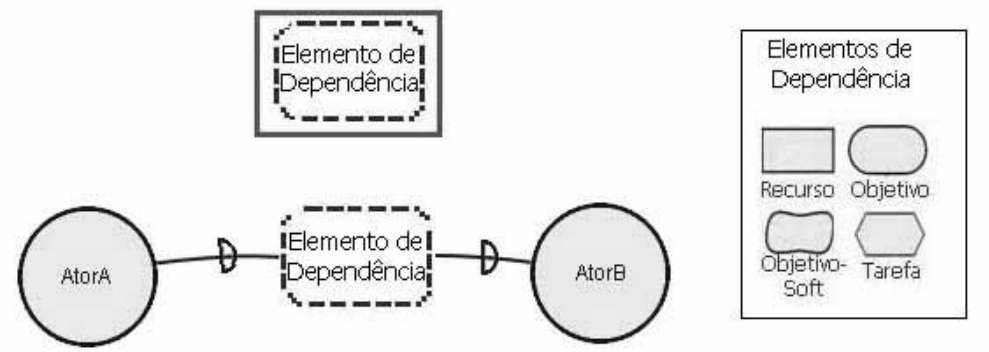


Figura 82 - Regra OCL: não deve haver elementos de dependência sem ligação no modelo.

Para o nó que corresponde a um ator no modelo:

Tabela 19 - Regra OCL: não deve haver atores sem ligação no modelo.

```
self.actors -> forAll(actor | (self.links -> exists(link |
link.source=actor or link.target=actor)) or (self.associations ->
exists(association | association.source=actor or
association.target=actor)))
```

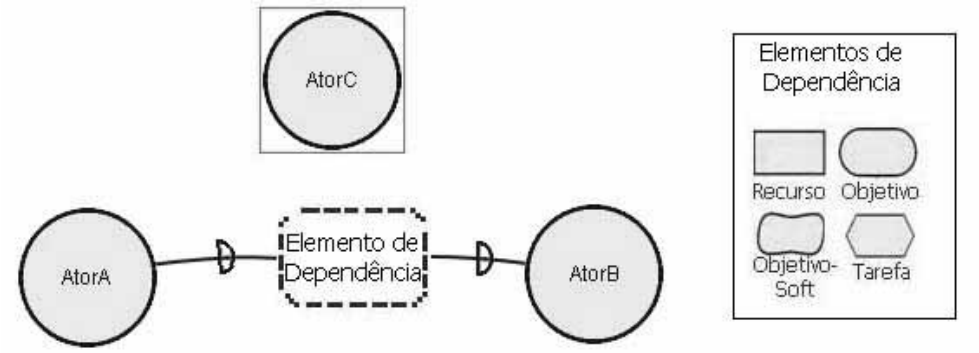


Figura 83 - Regra OCL: não deve haver atores sem ligação no modelo.

(xiii) Objetivos-*soft* só podem ser refinados usando-se *links* de contribuição. O *link* é usado de qualquer elemento para um Objetivo-soft.

Tabela 20 - Regra OCL: uso de links de contribuição.

```
self.contributionsLinks -> forAll(contribution |
contribution.target.type=ElementType::SOFTGOAL)
```

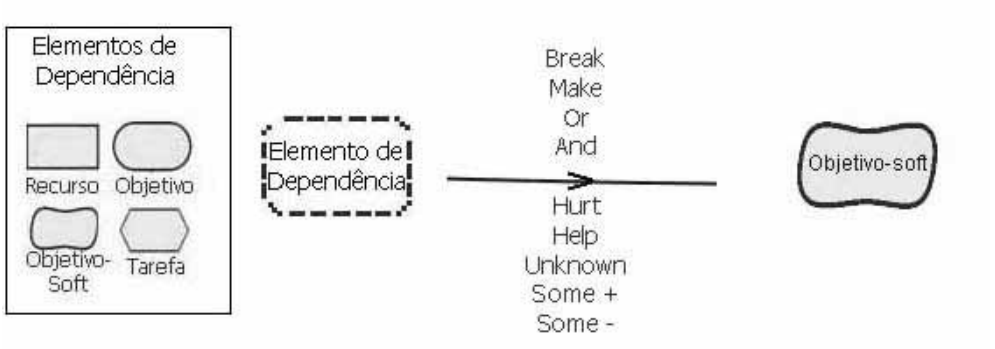


Figura 84 - Regra OCL: uso de links de contribuição.

Uma vez que o Modelo de Mapeamento foi finalizado, ele foi usado para originar o modelo gerador. Este irá gerar o código executável da ferramenta modelada.

4.3.6 Criação do Modelo de Geração

O Modelo de Geração, como apresentado na seção 4.2.5, é usado através do GMF para produzir o código executável da ferramenta. Esse modelo contém algumas características que podem ser modificadas a fim de produzir comportamentos específicos para a ferramenta. As propriedades que foram mudadas no modelo gerado pelo *framework* GMF, foram as que configuram o acionamento da execução das restrições definidas em OCL no modelo de mapeamento.

Foi necessário realizar modificações de três propriedades na opção *Gen Diagram ModelEditPart* do modelo (vide *Figura 85*): (a) a propriedade que habilita a verificação dos modelos (*Validation Enable*), (b) a propriedade que configura a verificação automática de consistência durante a construção dos modelos através do uso da ferramenta (*Live Validation UI Feedback*), e (c) a propriedade que indica se a prioridade da verificação é baixa, média ou alta (*Validation Provider Priority* definida para prioridade alta). Essa modificação significa que a ferramenta permitirá apenas a criação de modelos válidos segundo as restrições definidas nesta seção.

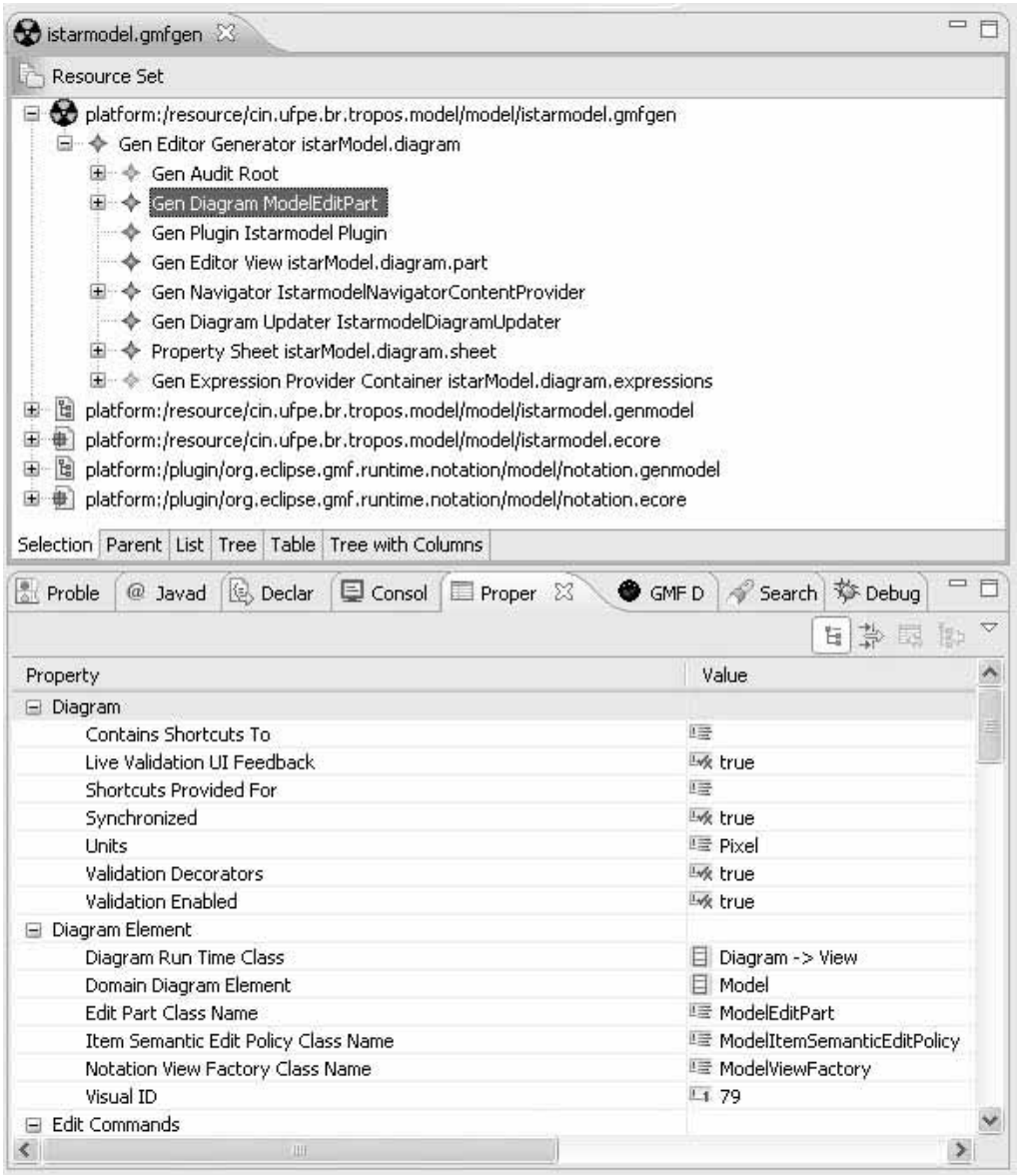


Figura 85 - Modelo de geração para a ferramenta de modelagem i*.

Na *Figura 85* temos a visualização do modelo de geração para a ferramenta criada. A partir do modelo de geração utilizamos mais uma vez o *framework* GMF para gerar, enfim, a versão executável da ferramenta - opção *Generate diagram editor* do GMF *Dashboard*. O código é gerado criando-se um novo projeto (*cin.ufpe.br.tropos.model.diagram*), que quando executado possibilita a criação de modelos i*.

Na *Figura 86* temos uma visão geral da ferramenta IStar Tool. No menu direito temos a paleta com as opções criadas no modelo gráfico, no centro temos a área onde a criação dos modelos é executada, no lado esquerdo temos acima o *workspace* - espaço de trabalho onde são criados os projetos e arquivos, e abaixo temos um esboço do modelo que está sendo construído.

No próximo capítulo utilizaremos como cenários a serem aplicados na ferramenta Istar Tool os exemplos de diagramas disponíveis no guia do i* [Grau 2008].

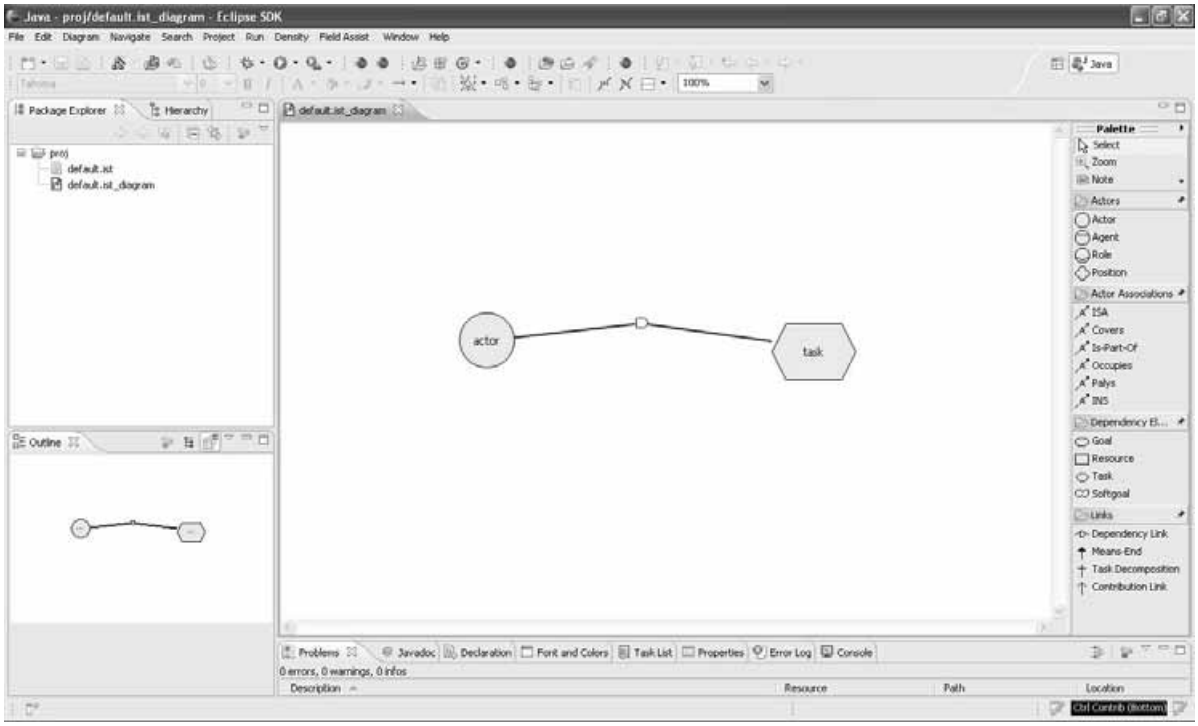


Figura 86 - Visual da ferramenta de modelagem i*.

4.4 Considerações

A criação e implementação de algo a partir de modelos é a base do Desenvolvimento Orientado a Modelos [MDA 2008]. Essa abordagem vem ganhando cada vez mais aceitação, pois dentre outros fatores: (i) facilita o projeto em todos os níveis arquiteturais proporcionando simplicidade e agilidade no desenvolvimento, e (ii) permite a análise do que foi gerado.

O desenvolvimento da ferramenta IStar Tool, apresentada neste capítulo, foi baseado no uso do *framework* GMF (*Graphical Modelling Framework*) [GMF 2008], que provê um processo para desenvolvimento baseado em modelos. A criação dos modelos foi fundamentada no estudo detalhado da linguagem i* - apresentado no capítulo anterior, seção 3.1.1 - e no estudo de guias e boas práticas - apresentado na seção 4.3.1. Os modelo criados seguindo o GMF envolveram: (i) a definição do meta-

modelo (modelo de domínio) para diagramas i*, (ii) a definição dos modelos gráfico e ferramental, (iii) a definição do modelo de mapeamento que combina as definições dos modelos de domínio, gráfico e ferramental, e (iv) a criação do modelo de geração que gera a versão executável (código Java) da ferramenta IStar Tool.

Um ponto importante na criação destes modelos foi a definição, no modelo de mapeamento, das restrições sobre o uso da linguagem i* de acordo com os guias e boas práticas definidas em [Grau 2008] e com o catálogo de erros mais comuns levantados em [Santos 2008]. Nem todos os itens do catálogo de [Santos 2008] foram contemplados nesta dissertação pois alguns deles dependem da interpretação do conteúdo dos elementos. Como apresentado na seção 4.3.5 - no item *Restrições Definidas em OCL* - nossa preocupação é focar em um conjunto de regras básicas sobre o uso correto das ligações dos elementos na construção de modelos em i*.

A ferramenta IStar Tool é capaz de realizar a verificação dos erros mais comuns encontrados em modelos i*. Contudo, a verificação de alguns dos erros apresentados em [Grau 2008][Santos 2008] ainda não foi contemplada nesta versão, ficando seu tratamento para trabalhos futuros. A verificação de consistência dos modelos foi definida para ser realizada à medida que os modelos estão sendo criados. Portanto, a ferramenta IStar Tool só permite a criação de modelos válidos.

Capítulo

5 Exemplos de Uso

Este capítulo apresenta exemplos de uso da ferramenta apresentada e descrita no capíutlo 4 deste trabalho. Aqui serão aplicados cenários à ferramenta IStar Tool segundo exemplos disponíveis em [i Wiki 2008][Grau 2008][Santos 2008]. O objetivo, portanto é exemplificar o uso da ferramenta e verificar se as restrições definidas estão sendo executadas corretamente.*

5.1 Cenários de Uso da Ferramenta IStar Tool

Como já mencionado, o objetivo deste capítulo é apresentar a utilização da ferramenta IStar Tool a fim de que se possa verificar se seu funcionamento está de acordo com o que foi definido nos modelos criados e nas restrições elaboradas. Utilizaremos para tanto alguns dos cenários disponíveis no guia do i* [Grau 2008] colocado como anexo deste trabalho e utilizado como referência para a definição e a implementação que foram apresentadas no capítulo 4.

Abaixo seguem a descrição dos cenários utilizados, juntamente com uma representação utilizada no guia do i* [Grau 2008], e uma descrição do comportamento da ferramenta implementada associada a uma imagem do resultado.

5.1.1 Uso de Links de Associação

Cenário 1: *Como tratado no Guideline 4.1.2.7 de [Grau 2008] - Papéis, Agentes, e Posições são atores de diferentes tipos, que caracterizam diferentes níveis de abstração. Isto significa que um Papel, por exemplo, pode ser parte de um outro Papel, mas não pode ser parte de um Agente ou de uma Posição. E isto também é uma verdade para um Agente e uma Posição*.

O guia coloca os seguintes cenários como correto (1) e incorreto (2), respectivamente:

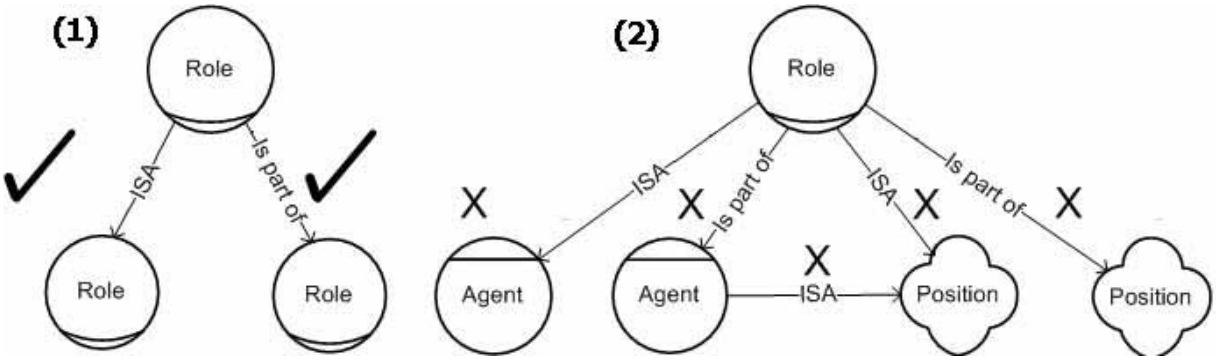


Figura 87 - Cenário 1: uso de links de associação entre atores segundo [Grau 2008].

Usando a ferramenta, IStar Tool, modelada e criada nesta dissertação tivemos o seguinte comportamento (*Figura 88* e *Figura 89*):

(i) A aplicação do cenário 1-(1) foi possível através da ferramenta (vide *Figura 88*).

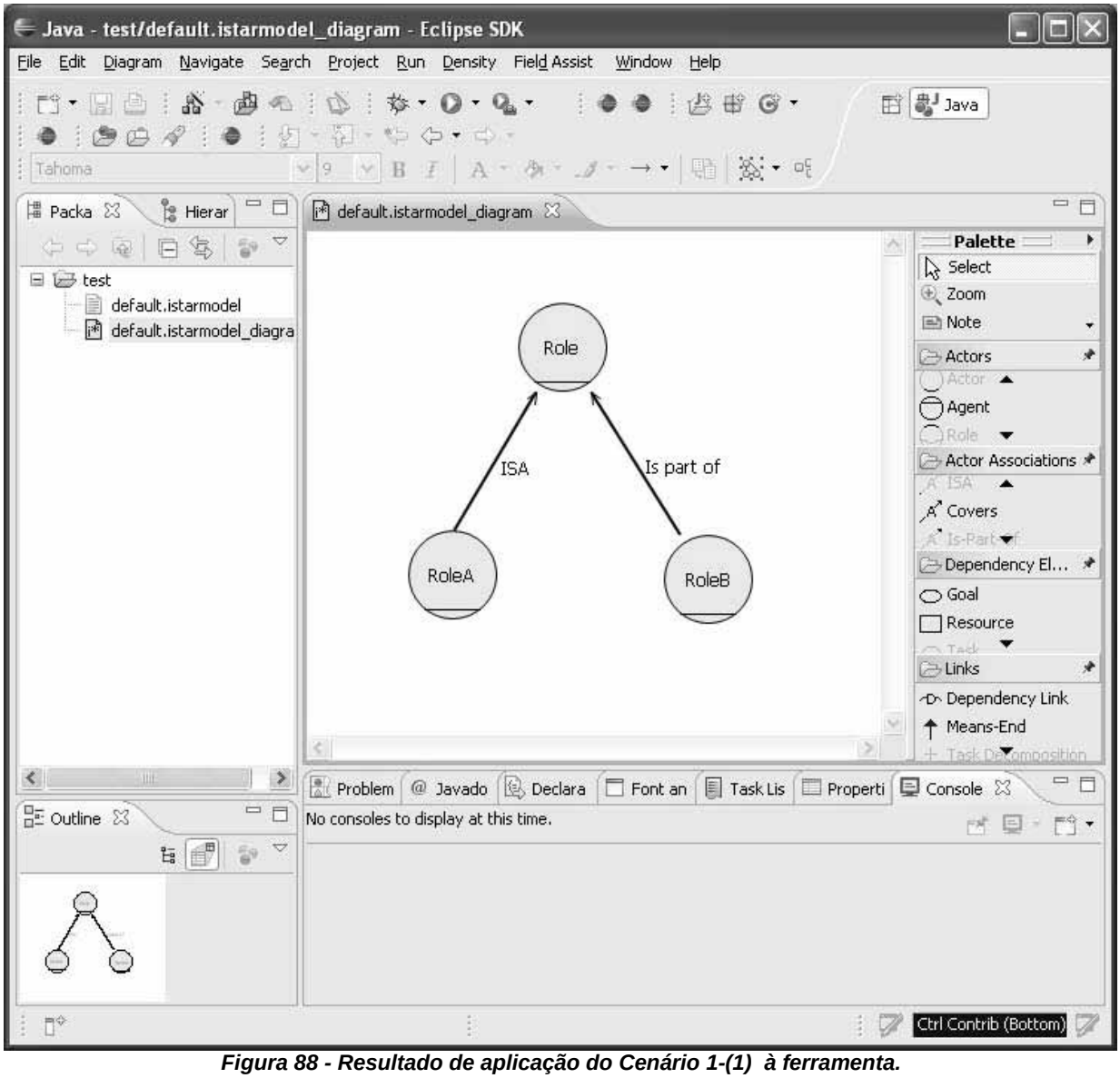


Figura 88 - Resultado de aplicação do Cenário 1-(1) à ferramenta.

(ii) A ferramenta não permitiu a construção do cenário 1-(2) - vide *Figura 89* - e indicou a seguinte mensagem de erro: "*ERROR: It is not possible to connect the elements. This type of Association Link is only used to connect actors of the same type.*". A mensagem de erro é dada à medida que se tenta realizar a conexão. A ferramenta não só não permite que a associação seja realizada, como também exibe esta mensagem de erro.

(iii) A outra mensagem que pode ser visualizada é referente a uma boa prática indicada no guia do i* - e citada no capítulo 4 deste trabalho, que diz que não devem existir nós (atores ou elementos de dependência) soltos no modelo sem qualquer ligação. A mensagem apareceu, pois uma vez que a ferramenta não permitiu a construção do cenário 2, os atores não ficaram associados a nenhum outro nó. A mensagem dada é: "*Warning: There is an actor with no connection with other elements.*".

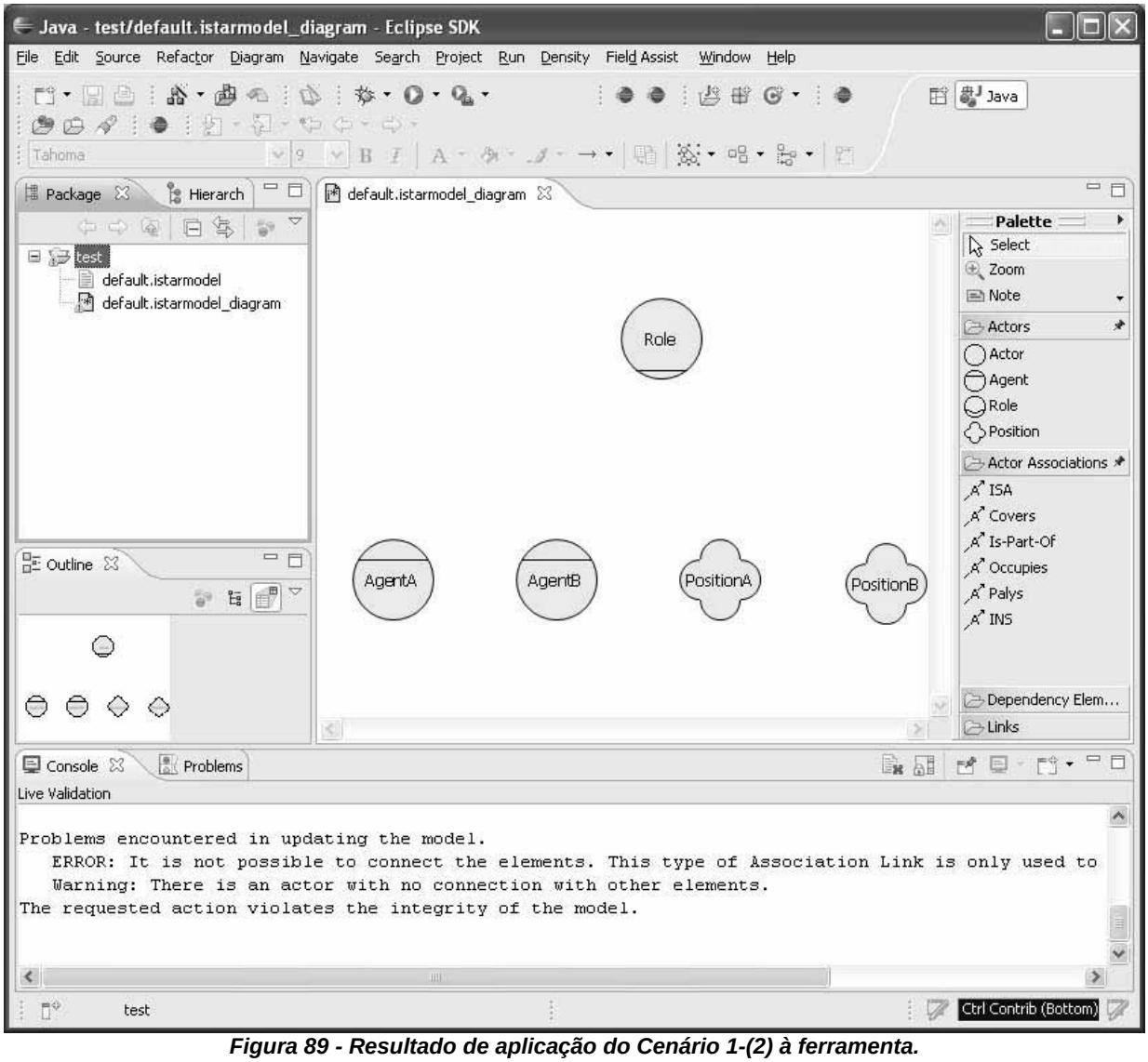


Figura 89 - Resultado de aplicação do Cenário 1-(2) à ferramenta.

A aplicação dos cenários acima está de acordo com as boas práticas do guia do i* [Grau 2008] e o cenário 1-(2) está de acordo com o conteúdo da Tabela I do catálogo de [Santos 2008], Anexo A desta dissertação. É importante observar que [Santos 2008] não apresenta em seu catálogo erros relativos ao uso das associações entre atores, ele apenas indica que um ator não pode estar sem ligação em um modelo i*. Cenários relativos ao uso correto de link de associação tem como referência os guias de [Grau 2008].

Cenário 2: *Como tratado no Guideline 4.2 de [Grau 2008] - As relações entre atores são descritas por link de associação entre atores*.

Utilizamos a ferramenta e tentamos realizar a associação de um ator a um outro elemento, no nosso exemplo, a um objetivo. A ferramenta não permitiu que a ligação fosse realizada (*Figura 90*). Ao tentar ligar o *AtorA* ao *Goal* a ferramenta indicou a impossibilidade da ligação ocorrer.

Como a ligação não pode ser realizada, o modelo ficou com dois elementos sem ligação, um ator e um objetivo, e portanto, exibiu as seguintes mensagens: "*Problems encountered in updating the model. Warning: There is a dependency element with no connection with other elements. Warning: There is an actor with no connection with other elements.".*

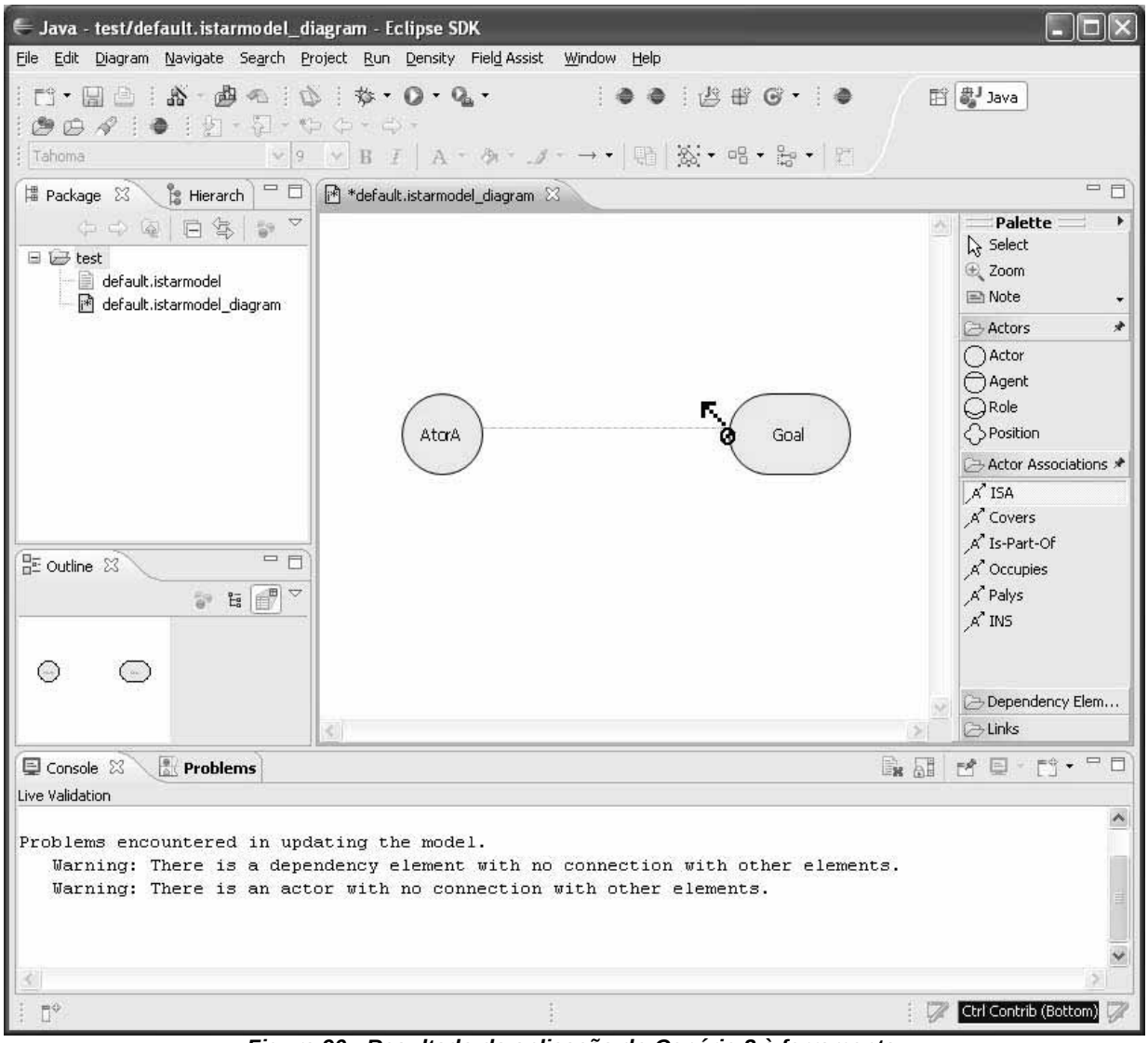


Figura 90 - Resultado da aplicação do Cenário 2 à ferramenta.

Cenário 3: *Como tratado no Guideline 4.1.4* de *[Grau 2008] - Um Papel pode ser associado a um Agente ou a uma Posição usando-se os links de associação Plays e Covers, respectivamente. Um agente por sua vez pode ser associado a uma Posição usando o link de associação Occupies.*

O guia coloca o cenário da *Figura 91* como correto de ser aplicado:

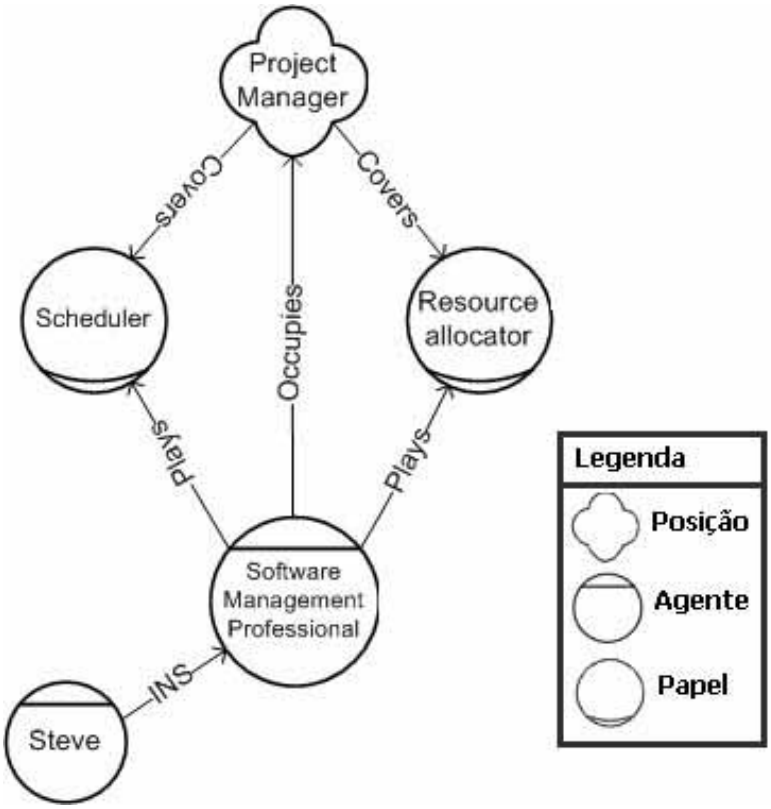


Figura 91 - Cenário 3: uso de links de associação de atores segundo [Grau 2008].

Usando a ferramenta modelada e criada neste trabalho obtivemos o comportamento esperado, ou seja, a ferramenta permitiu a ligação corretamente (*Figura 92*).

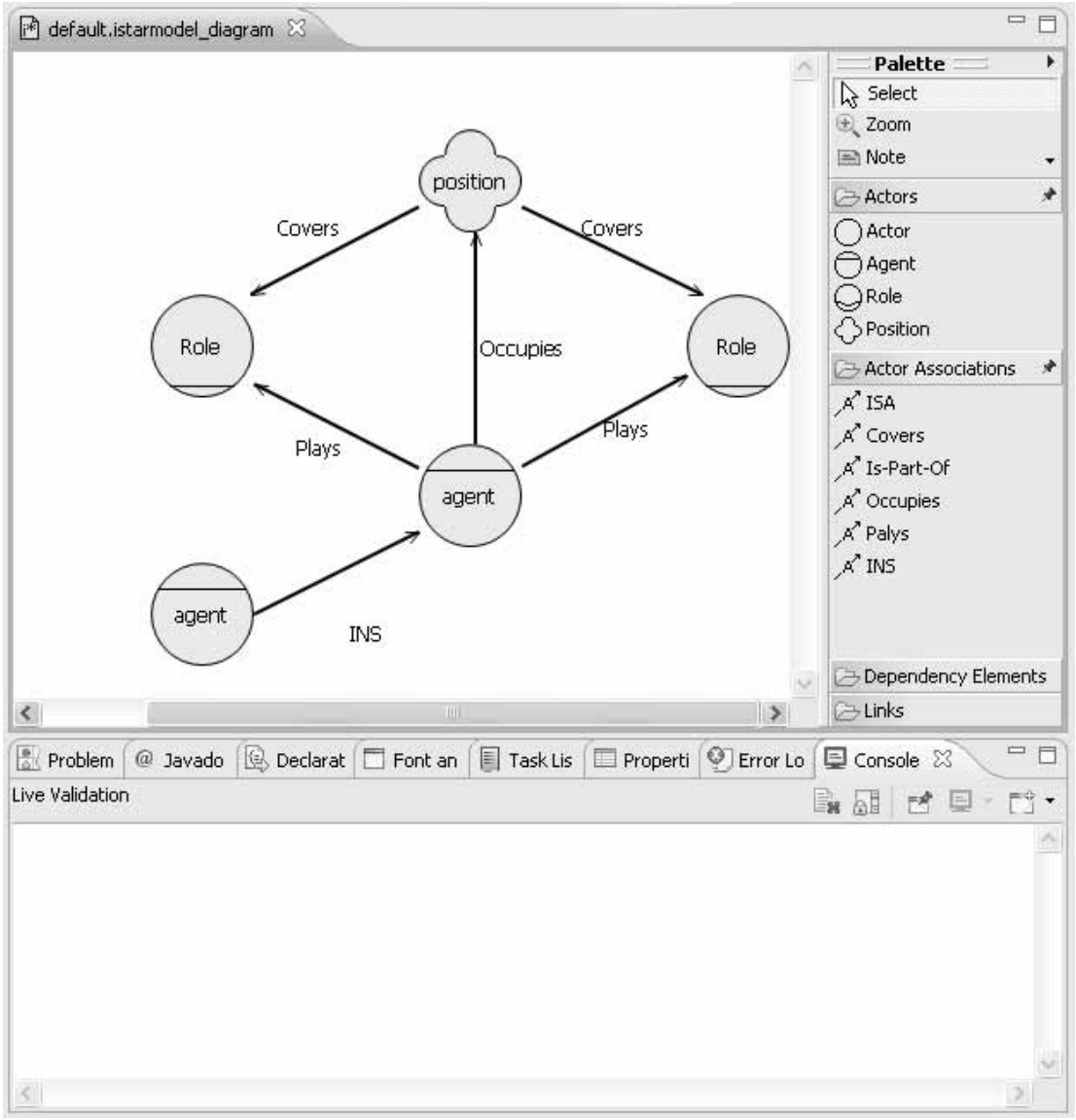


Figura 92 - Resultado de aplicação do Cenário 3 à ferramenta.

5.1.2 Reuso de Dependum

> **Cenário 4:** *De acordo com o apresentado no Guideline 4.3.10 de [Grau 2008] e na Tabela I do catálogo de [Santos 2008] - Um link de dependência deve ter um único elemento de ambos os lados do Dependum, ou seja, deve haver um único Depender e um único Dependee associados.*

O guia coloca como exemplo a *Figura 93* ilustrando, no primeiro cenário uma aplicação correta de *links* de dependência, e no segundo cenário uma aplicação ambígua de *links* de dependência. A figura mostra que [Grau 2008] o ator "*Patient*" depende do ator "*Government*" e do ator "*Other health service providers*" para atingir ou satisfazer o objetivo de "*Health Service Be Provided*". A forma como essa notação é construída no cenário (2) viola o conceito de Link de Dependência no i* porque isso torna

difícil para o projetista e para o analista entender e avaliar a satisfação do *Dependum* quando este é ligado a diferentes *Dependees*. Por conta da autonomia dos atores (*Dependees*) em i*, cada ator é avaliado separadamente em termos de suas contribuições para o preenchimento e satisfação do *Dependum*.

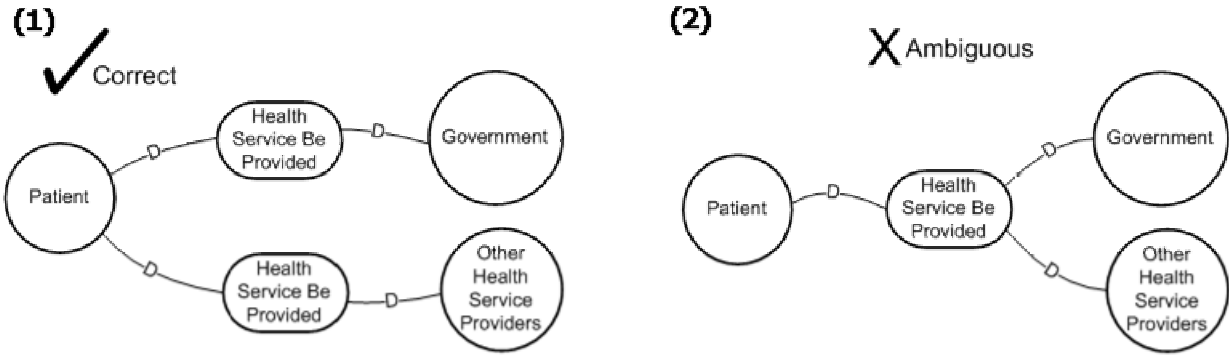


Figura 93 - Cenários 4: uso de um Dependum em mais de uma relação segundo [Grau 2008].

Usando a ferramenta apresentada nesta dissertação tivemos o seguinte comportamento (vide *Figura 94*): a ferramenta não permitiu a ligação entre o *Goal* e o *AtorC*, e indicou a seguinte mensagem: "*ERROR: It is not possible to use a dependum in more than one relationship in the diagram*".

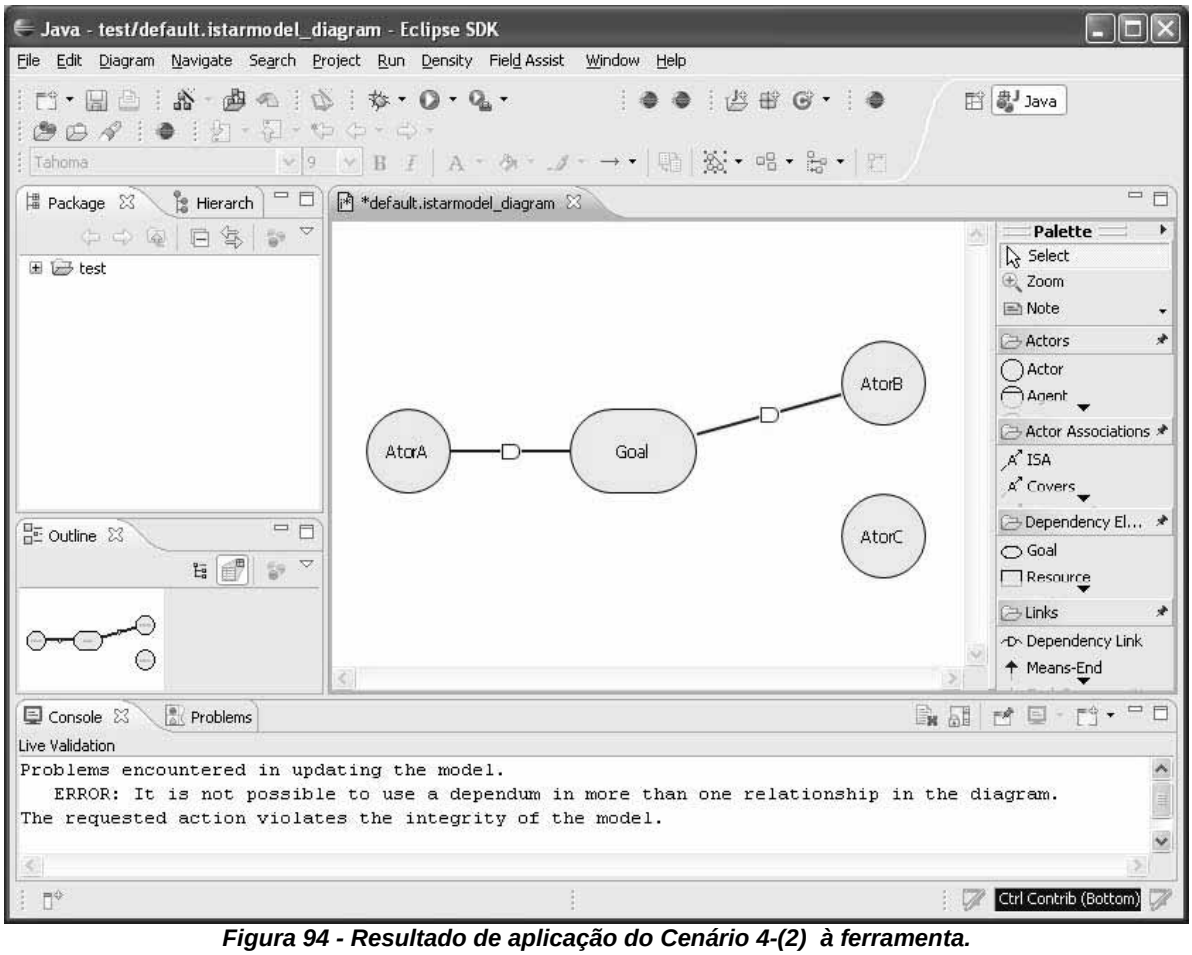


Figura 94 - Resultado de aplicação do Cenário 4-(2) à ferramenta.

A aplicação do cenário considerado mais indicado segundo [Grau 2008] para representar esse link de dependência conforme indicado na *Figura 93 - (1)*, apresentou o seguinte comportamento da ferramenta (*Figura 95*). Este comportamento também está de acordo com o conteúdo da Tabela I do catálogo de [Santos 2008], Anexo A desta dissertação.

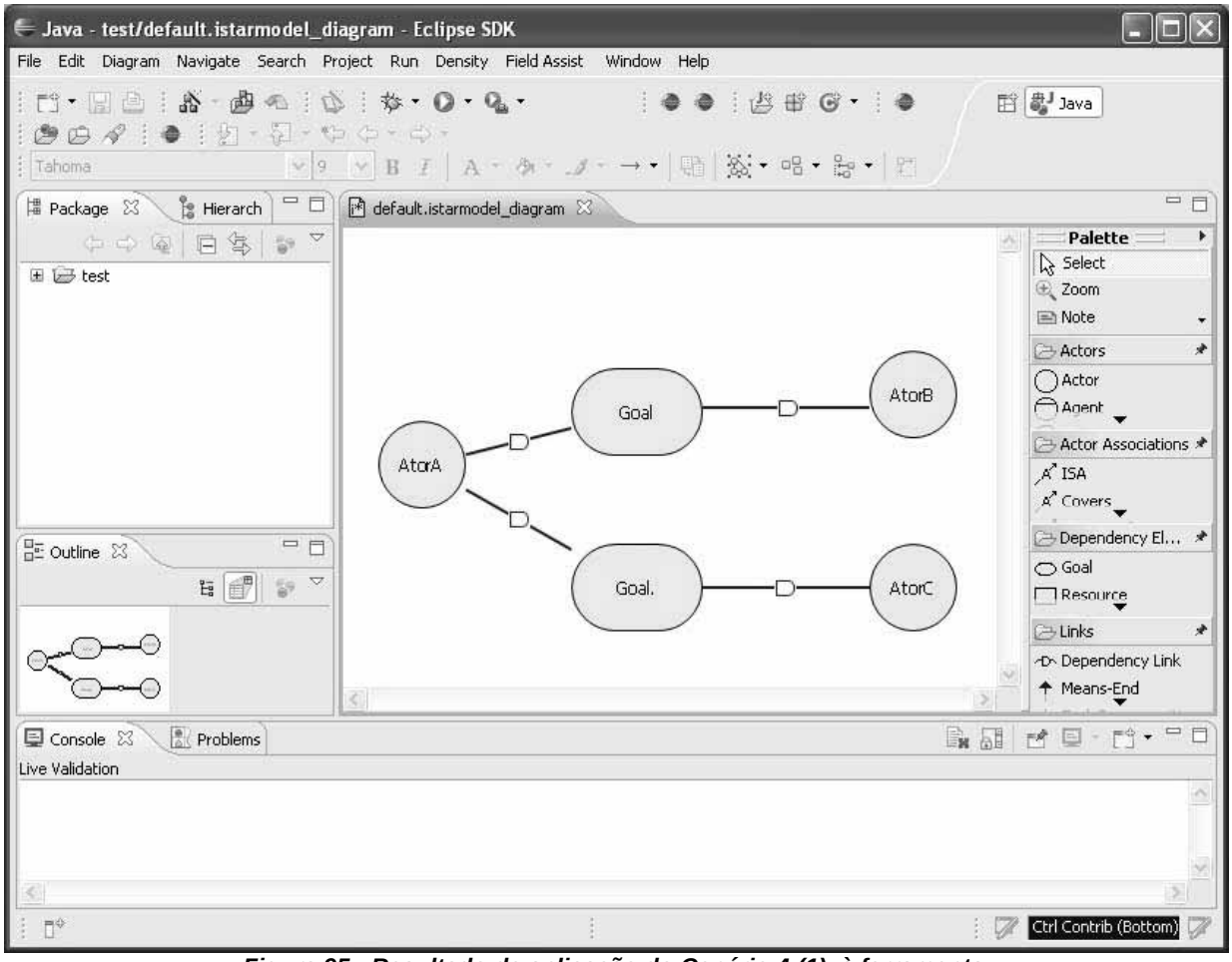


Figura 95 - Resultado de aplicação do Cenário 4-(1) à ferramenta.

5.1.3 Uso de Link de Dependência

> **Cenário 5:** *De acordo com o apresentado no Guideline 4.3.7 de [Grau 2008] e na Tabela VII do catálogo de [Santos 2008] - Deve-se usar o símbolo "D" como notação para denotar um Link de Dependência. Não usar outros símbolos tais como seta, que pode ser confundido com outros tipos de links.*

O guia coloca como exemplo a *Figura 96* ilustrando, uma aplicação correta da representação de links de dependência, e uma aplicação incorreta que pode confundir o link de dependência com outros links no modelo.

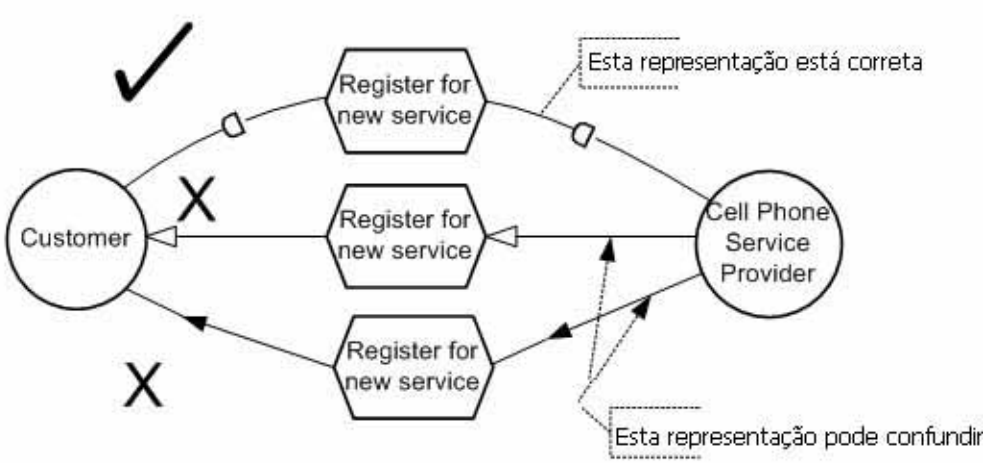


Figura 96 - Cenário 5: uso de um Dependum em mais de uma relação segundo [Grau 2008].

Usando a ferramenta IStar Tool (vide *Figura 97*) podemos notar que a representação usada para link de dependência está de acordo com o guia [Grau 2008] e está de acordo também com o conteúdo da Tabela VII do catálogo de [Santos 2008], Anexo A desta dissertação.

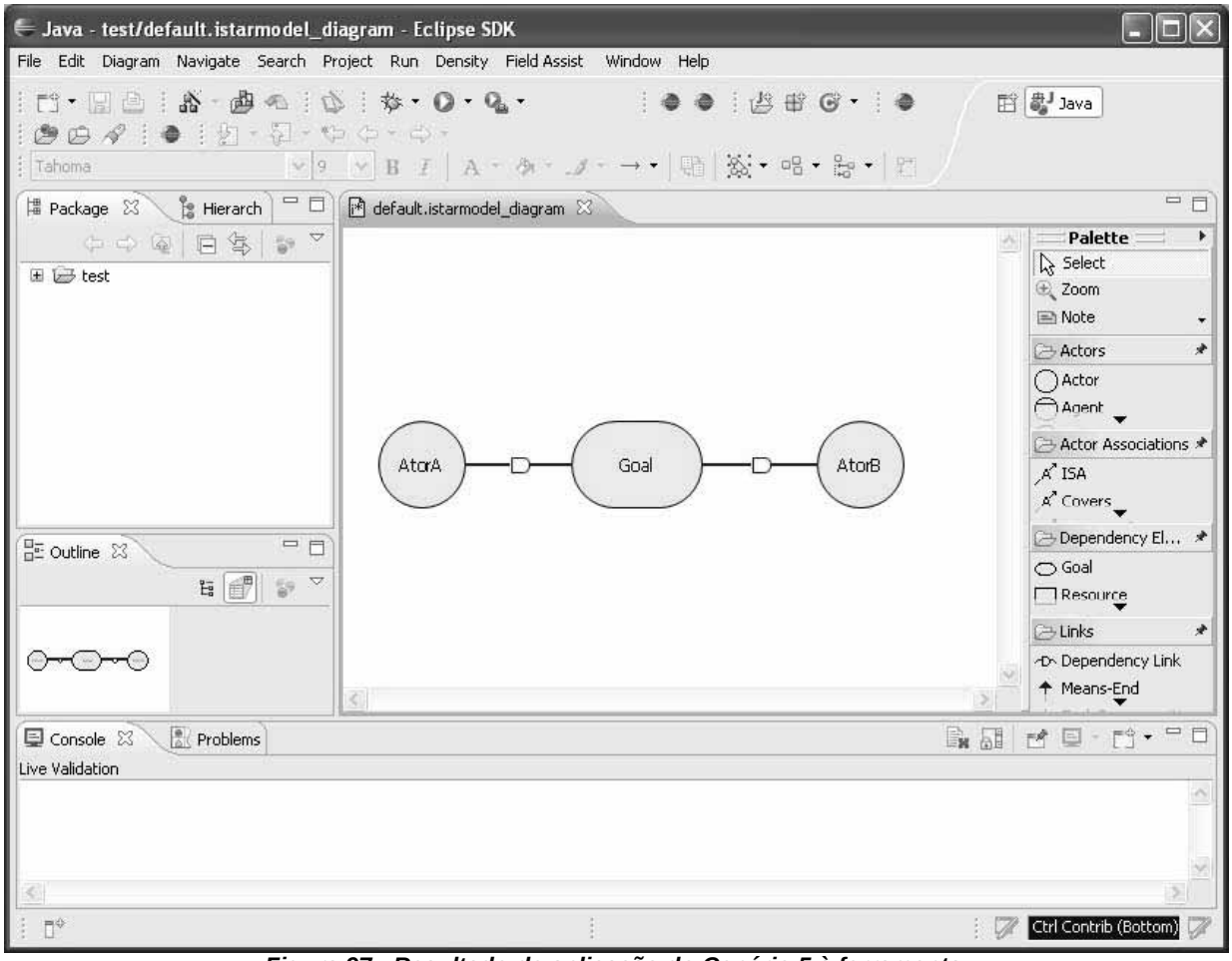


Figura 97 - Resultado de aplicação do Cenário 5 à ferramenta.

Cenário 6: *De acordo com o apresentado no Guideline 4.3.11 de [Grau 2008] e na Tabela IX do catálogo de [Santos 2008] - Não deve ser usado um Link de Dependência entre dois atores sem mostrar o Dependum.*

Usando a ferramenta IStar Tool (vide *Figura 98*) podemos notar que a representação usada para link de dependência está de acordo com o guia [Grau 2008] e está de acordo também com o conteúdo da Tabela IX do catálogo de [Santos 2008], Anexo A desta dissertação. A ferramenta não apenas não permitiu a criação do link de dependência, como também exibiu a seguinte mensagem de erro: "*ERROR: It is note possible to create a dependency link between these nodes.*".

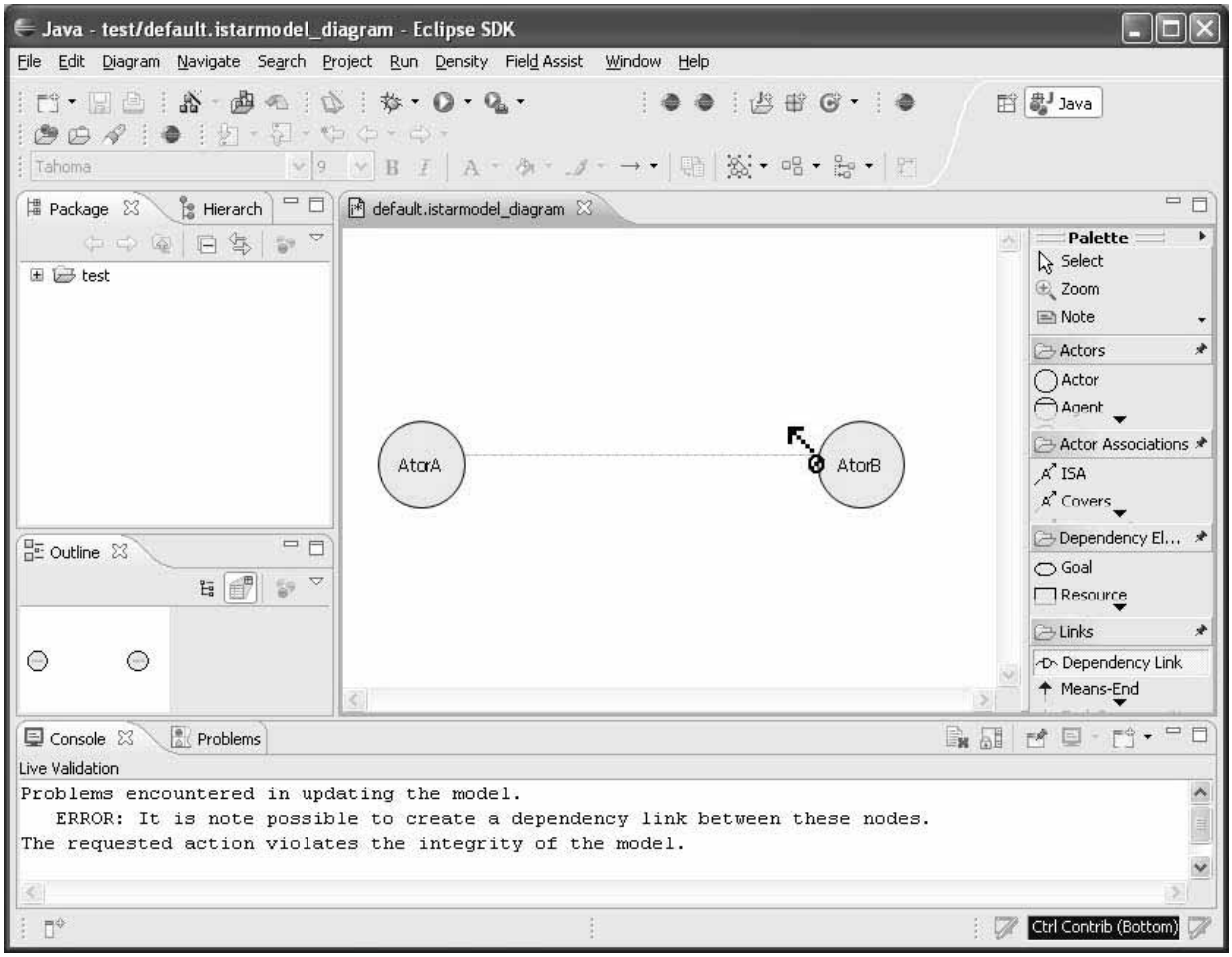


Figura 98 - Resultado de aplicação do Cenário 6 à ferramenta.

Cenário 7: *De acordo com o apresentado no Guideline 4.3.8 de [Grau 2008] e na Tabela VIII do catálogo de [Santos 2008] - Não deve ser usado um Link de Dependência dentro da fronteira de um ator.*

Usando a ferramenta IStar Tool (vide *Figura 99*) podemos notar que ao tentar utilizar a criação do link de dependência dentro da fronteira do ator a ferramenta não permite que a ligação seja feita.

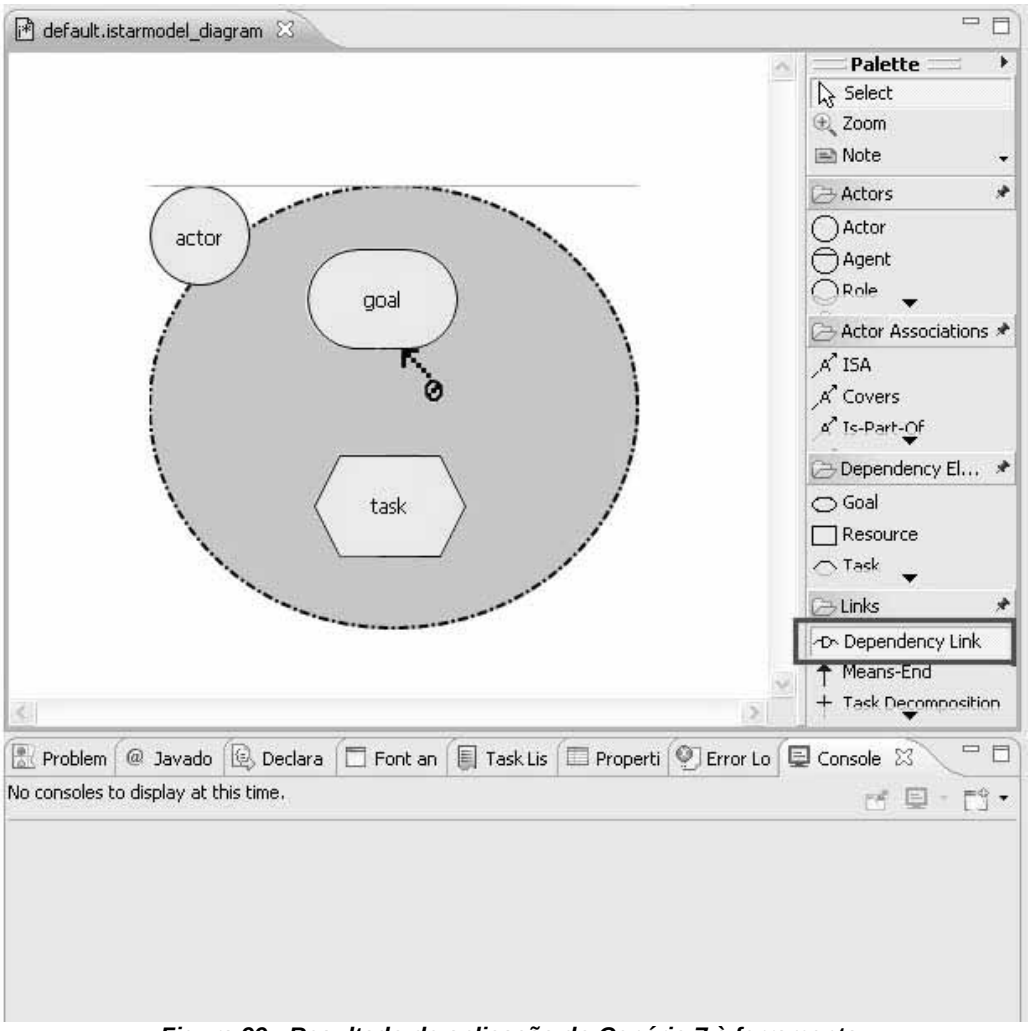


Figura 99 - Resultado de aplicação do Cenário 7 à ferramenta.

5.1.4 Uso da Fronteira de um Ator

Cenário 8: *De acordo com o apresentado no Guideline 5.1.3 - Deve-se usar a imagem convencional (um círculo) para representação da fronteira de um ator.*

De acordo com o modelado e desenvolvido na ferramenta apresentada neste trabalho temos a fronteira do ator representado de acordo com o *Guideline 5.1.3* como pode ser observado na *Figura 100*.

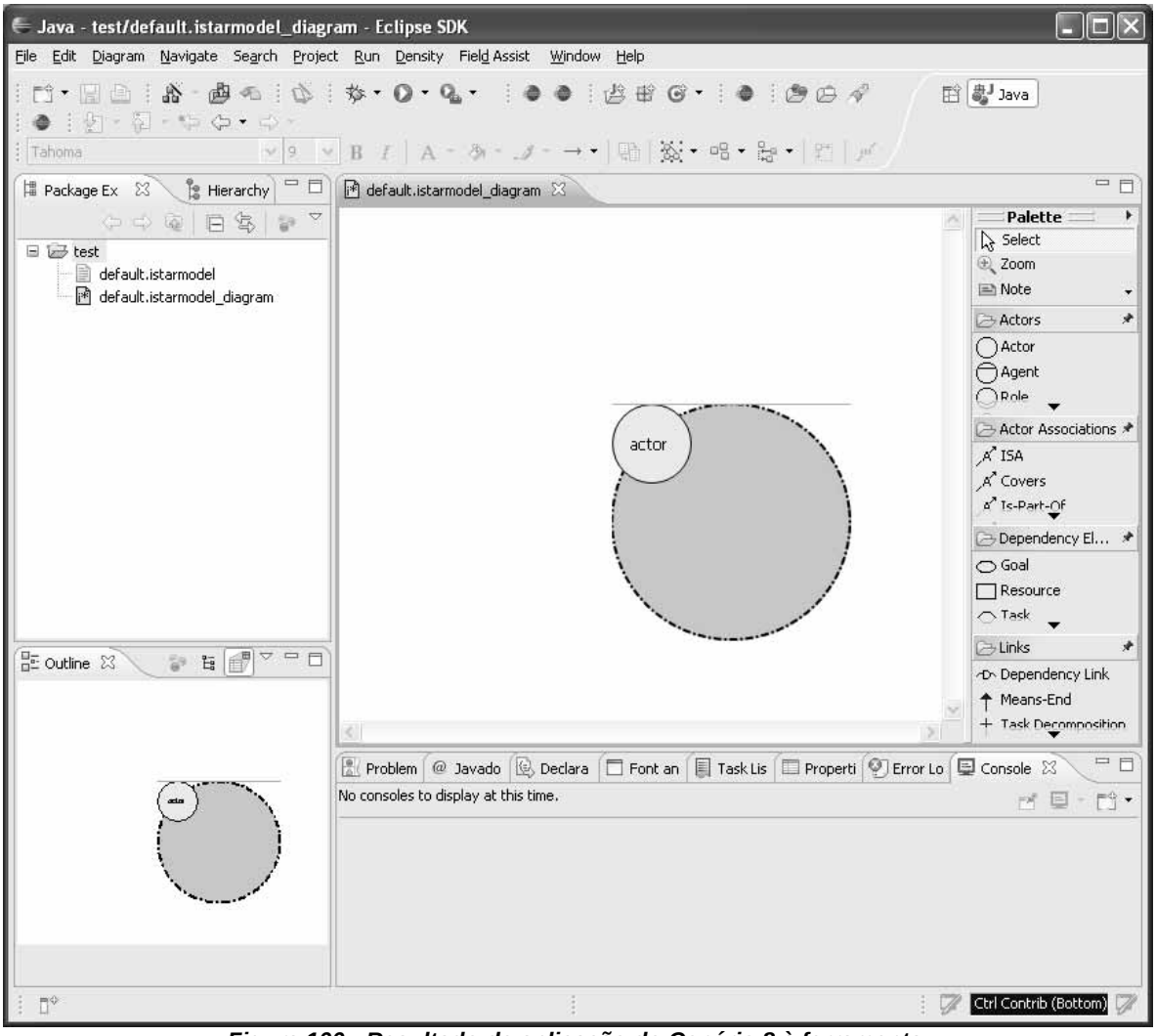


Figura 100 - Resultado de aplicação do Cenário 8 à ferramenta.

Cenário 9: *De acordo com o apresentado no Guideline 4.1.5 e de [Grau 2008] e na Tabela III do catálogo de [Santos 2008] - Não se deve incluir um ator dentro da fronteira de outro ator.*

Aplicando o cenário na ferramenta apresentada neste trabalho, o que acontece é que a ferramenta não permite a criação de um ator dentro da fronteira de outro ator. Primeiro selecionamos a opção na paleta de criação do ator e depois tentamos clicar dentro da fronteira do ator, já criado, para incluir o novo ator (vide *Figura 101*).

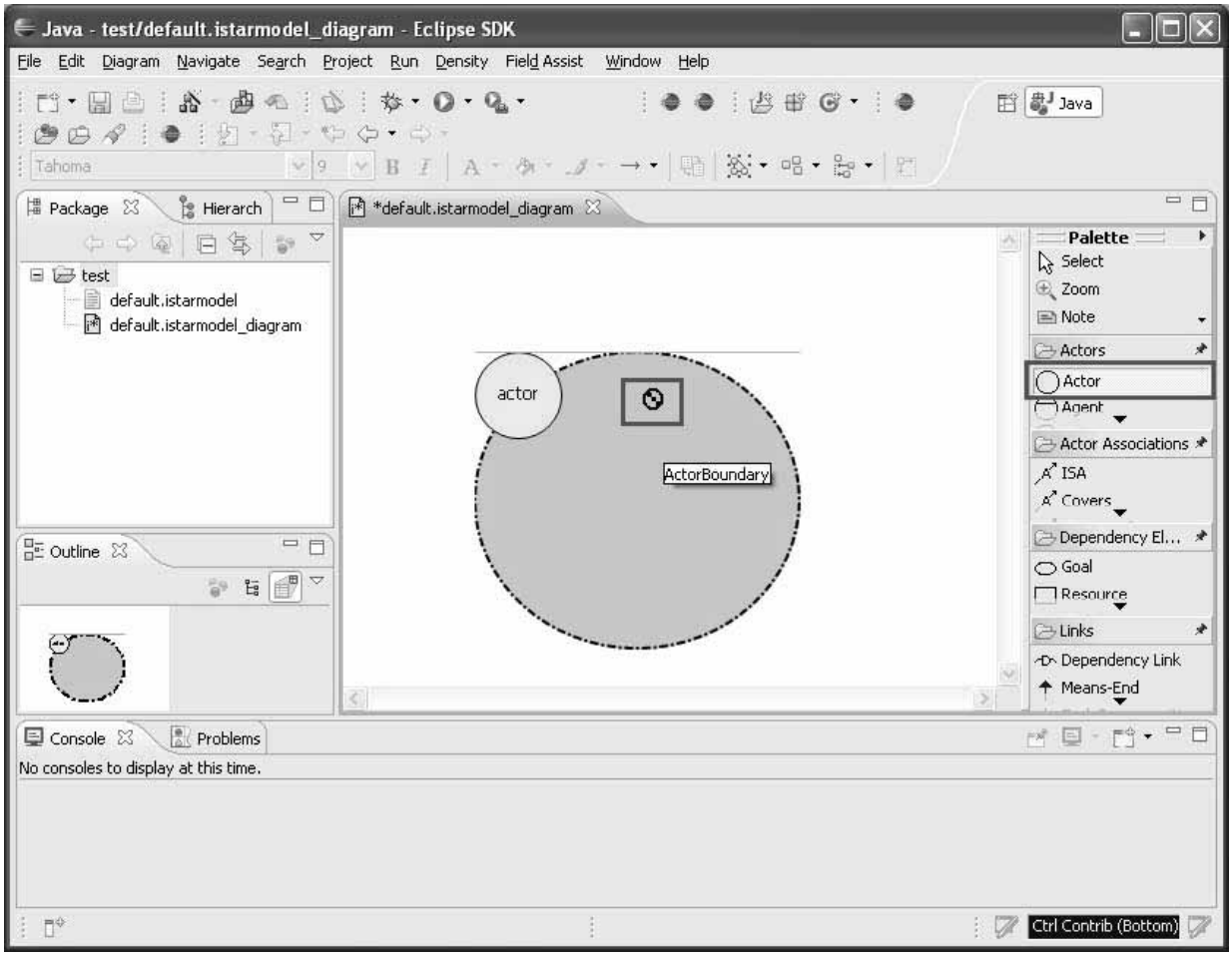


Figura 101 - Resultado de aplicação do Cenário 9 à ferramenta.

5.1.5 Uso do Link Meio-fim

> **Cenário 10:** *De acordo com o apresentado nos Guidelines 5.2.1.7 e 5.3, e na Tabela II do catálogo de [Santos 2008] - Um objetivo só pode ser decomposto usando-se a ligação de meio-fim. O link meio-fim é um tipo de relação que indica um fim (Objetivo) e os meios (Tarefas) para atingir este objetivo.*

O guia coloca como exemplo a *Figura 102*. De acordo com o modelado e desenvolvido na ferramenta IStar Tool neste trabalho, observamos na *Figura 103* que somente a construção correta do link meio-fim é possível. O uso do link meio-fim, assim como o uso dos *links* de decomposição de tarefas e de contribuição, na ferramenta está restrito às regras em OCL definidas no capítulo anterior. Portanto, tais *links* serão criados apenas se a origem e o destino da ligação obedecer às regras estabelecidas.

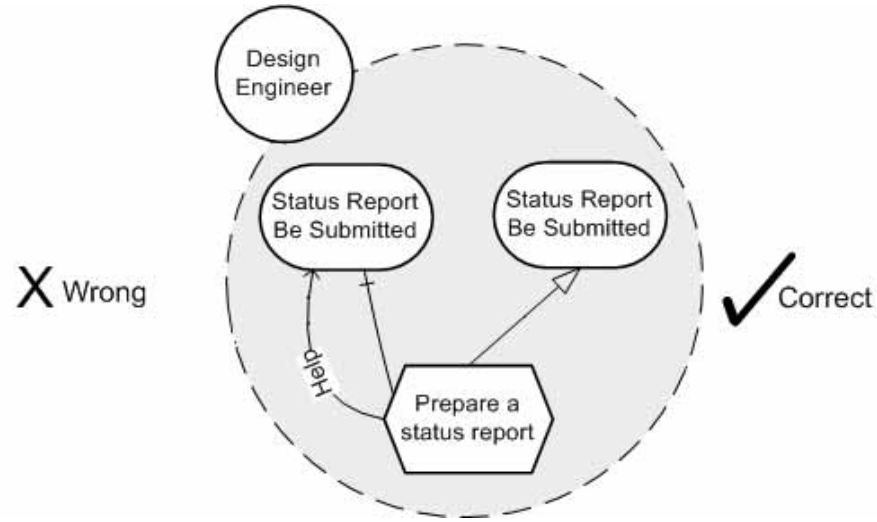


Figura 102 - Cenário 10: uso de link meio-fim segundo [Grau 2008].

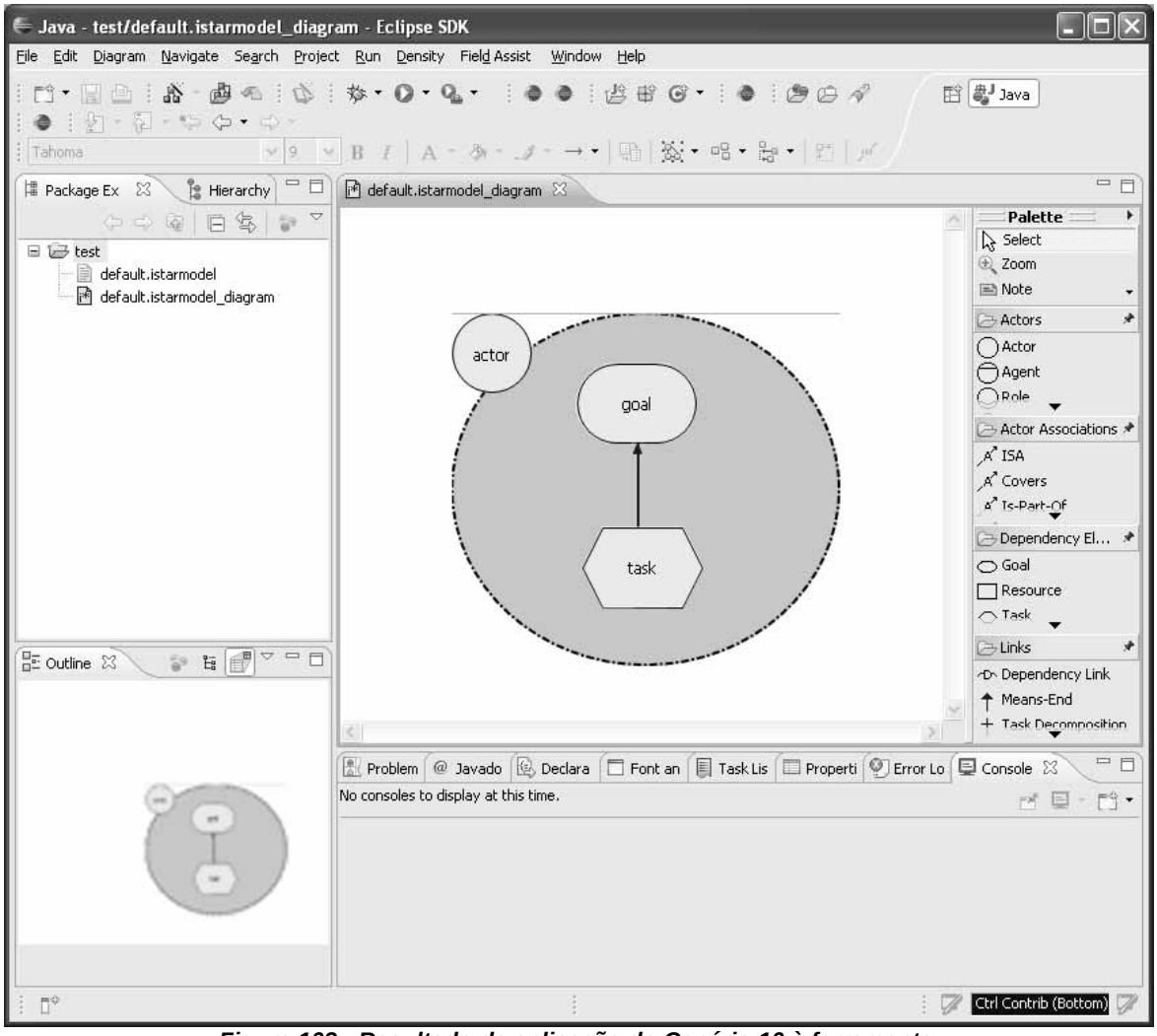


Figura 103 - Resultado de aplicação do Cenário 10 à ferramenta.

Cenário 11: *De acordo com o apresentado no Guideline 5.2.1.5 e Tabela XI do catálogo de [Santos 2008] - Não se deve misturar objetivos com tarefas em uma ligação meio-fim.*

Utilizando a ferramenta tentamos realizar uma ligação meio-fim de um objetivo para outro objetivo. A ferramenta não só não permitiu que a ligação fosse realizada como também indicou a seguinte mensagem de erro (vide *Figura 104*): "*ERROR: It is not possible to connect the elements. The Means-End link is only used to connect Task to a Goal*". Este comportamento está de acordo com o conteúdo da Tabela XI do catálogo de [Santos 2008], Anexo A desta dissertação.

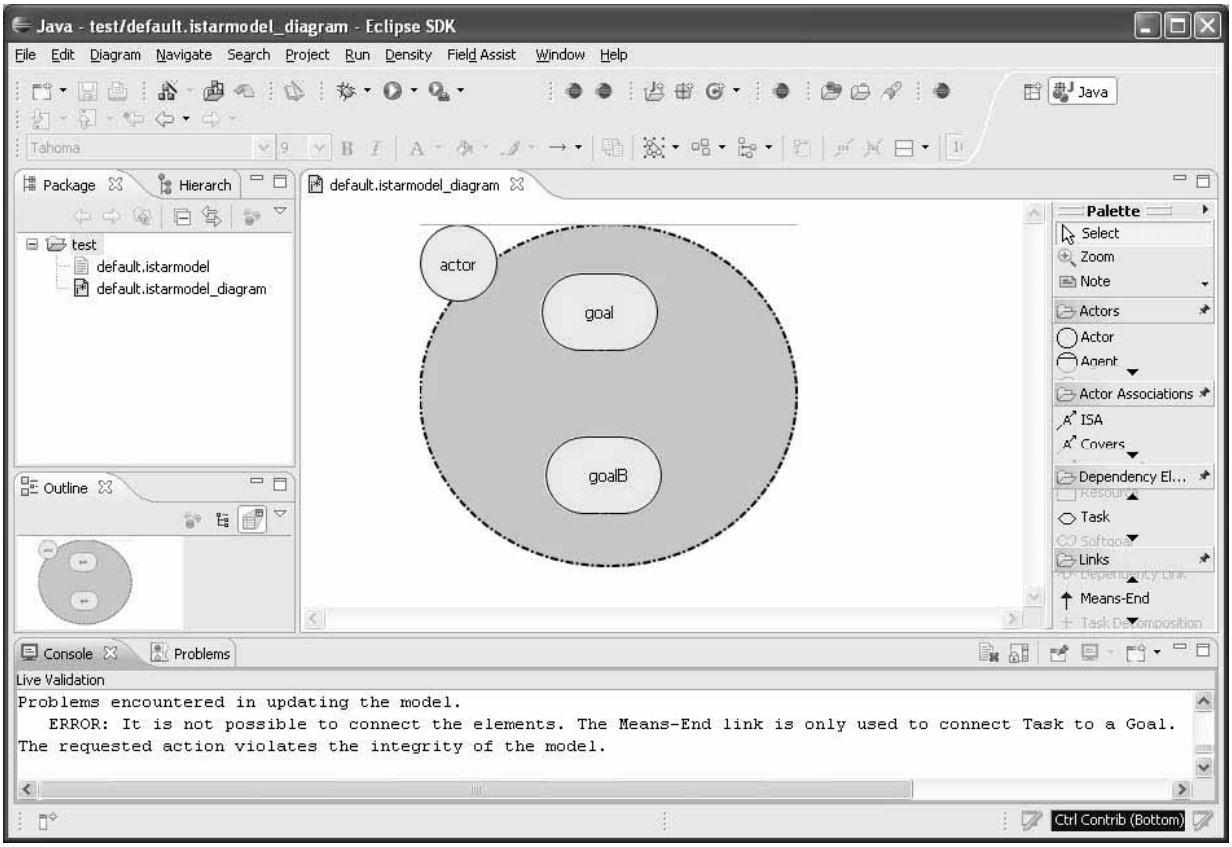


Figura 104 - Resultado de aplicação do Cenário 11 à ferramenta.

Cenário 12: *De acordo com o apresentado no catálogo de [Santos 2008], Tabela XIV - O link meio-fim não deve ser usado fora da fronteira do ator. A ligação meio-fim é feita somente entre elementos internos à fronteira do ator.*

Utilizando a ferramenta tentamos realizar uma ligação meio-fim de um objetivo para uma tarefa fora da fronteira de um ator. A ferramenta não permite que a ligação seja realizada como pode ser visualizado na *Figura 105*.

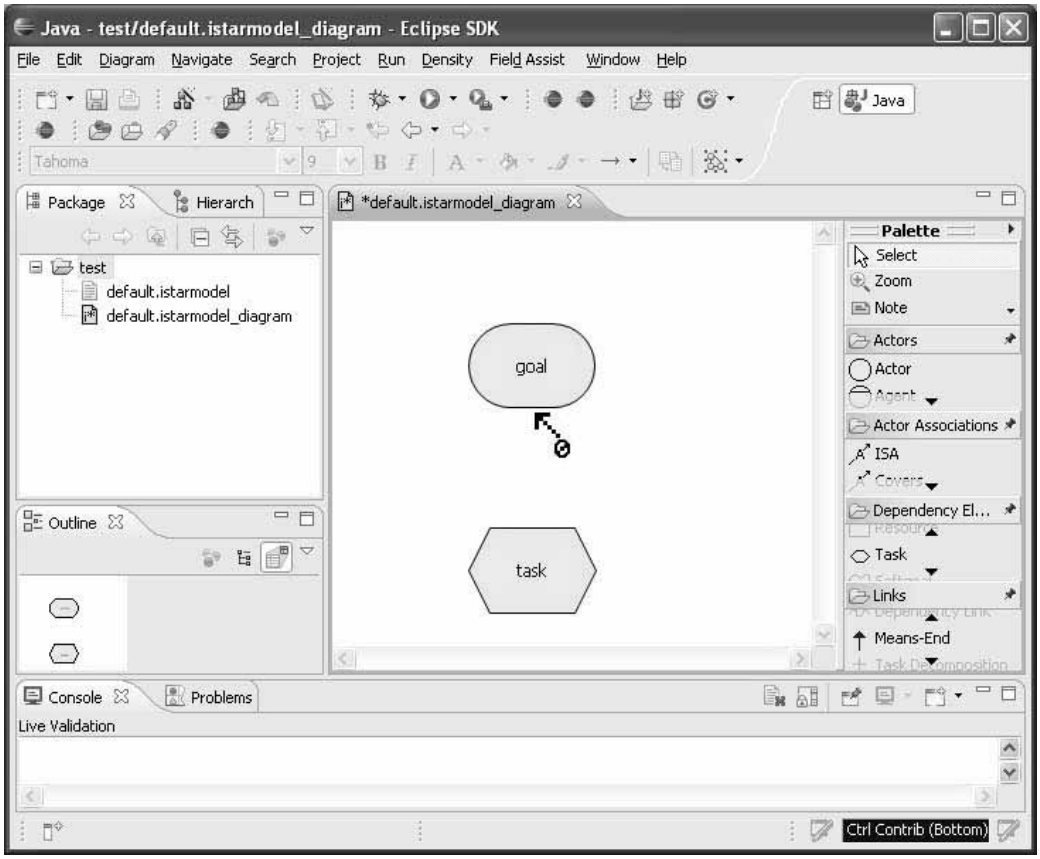


Figura 105 - Resultado de aplicação do Cenário 12 à ferramenta.

5.1.6 Uso do Link de Decomposição de Tarefas

Cenário 13: *De acordo com o apresentado nos Guidelines 4.2.4, e na Tabela XII do catálogo de [Santos 2008] - Uma tarefa pode ser decomposta utilizando-se o link de Decomposição de Tarefas. Uma tarefa pode ser decomposta em quarto tipos de elementos: objetivo-soft, tarefa, recurso, e ou objetivo. A tarefa pode ser decomposta em um ou mais elementos destes tipos.*

Utilizando a ferramenta IStar Tool, observamos na *Figura 106* o seu comportamento permitindo a criação de links de decomposição de tarefa dentro da fronteira de um ator.

Cenário 14: *De acordo com o apresentado na Tabela XIII do catálogo de [Santos 2008] - Não se deve usar decomposição de tarefas fora da fronteira do ator. A decomposição de tarefas é feita somente entre elementos internos às fronteiras dos atores.*

Utilizando a ferramenta tentamos realizar uma ligação de decomposição de tarefa, de um objetivo para uma tarefa, fora da fronteira de um ator. A ferramenta não permite que a ligação seja realizada como pode ser visualizado na *Figura 107*.

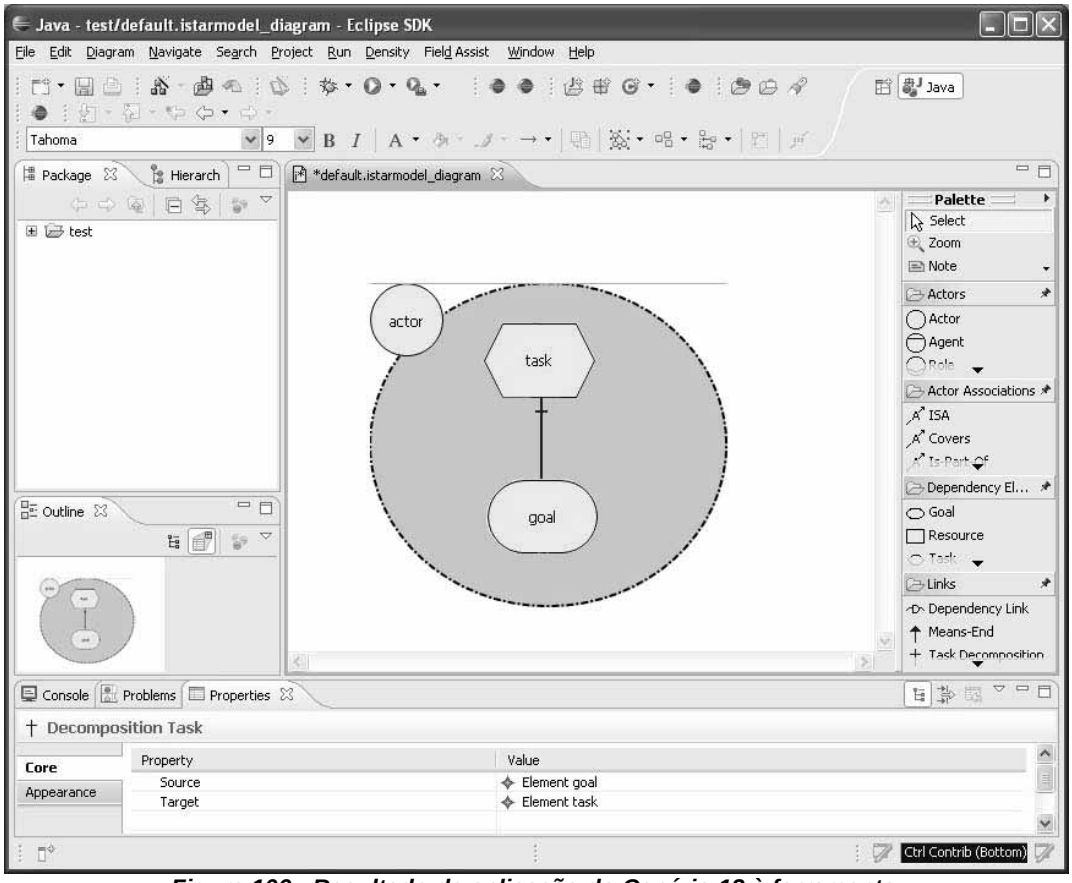


Figura 106 - Resultado de aplicação do Cenário 13 à ferramenta.

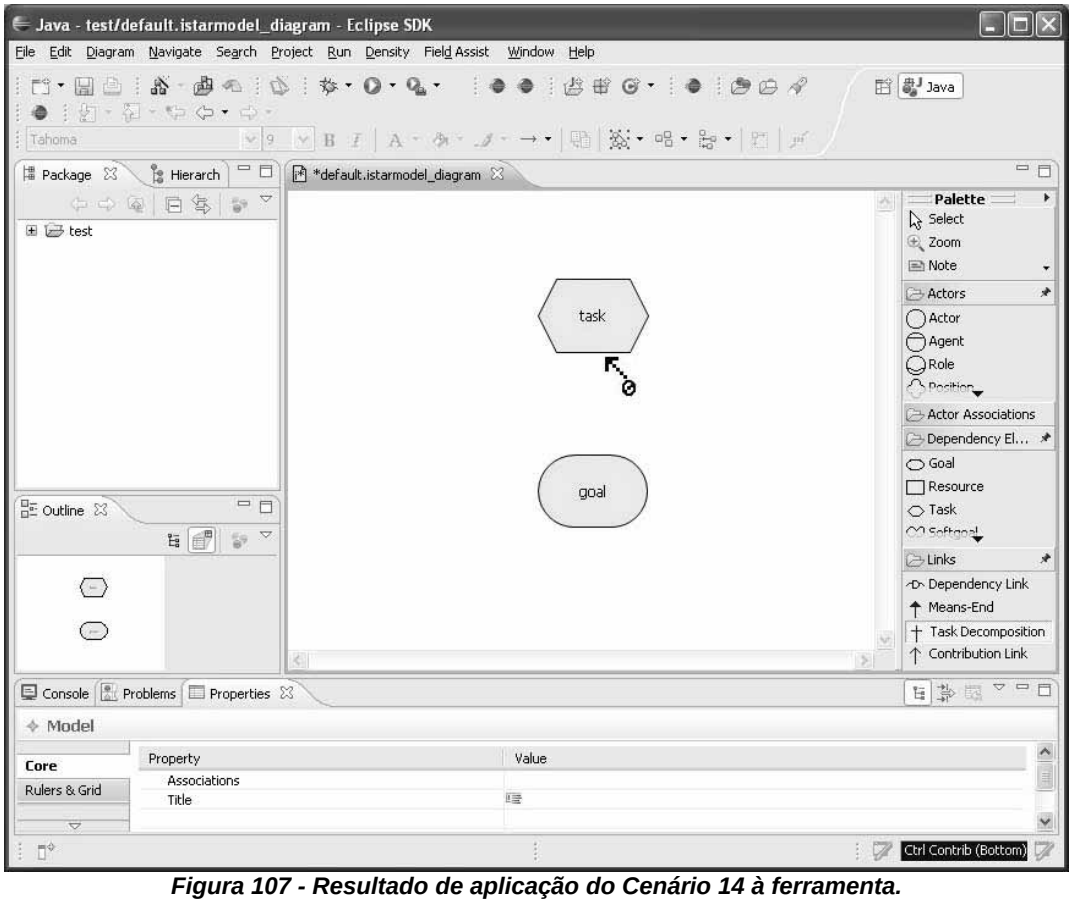


Figura 107 - Resultado de aplicação do Cenário 14 à ferramenta.

5.1.7 Uso do Link de Contribuição

Cenário 15: *De acordo com o apresentado nos Guidelines 4.2.5, e na Tabela X do catálogo de [Santos 2008] - Os links de contribuição devem ser utilizados para ligar elementos (objetivo, objetivo-soft, recurso, e/ou tarefa) a objetivos--soft. Os links de contribuição devem ser usados para modelar a forma como os elementos contribuem para a satisfação ou cumprimento do objetivo-soft.*

Utilizando a ferramenta IStar Tool, observamos nas Figuras 108 e 109 o seu comportamento permitindo a criação de links de contribuição dentro da fronteira de um ator. Primeiro, selecionamos a origem e o destino da contribuição, a ferramenta abre então um conjunto de opções para que seja selecionado o tipo da contribuição (vide *Figura 108*). Uma vez que o tipo da contribuição é selecionado a ferramenta realiza a ligação de acordo com o tipo de contribuição escolhido (vide *Figura 109*).

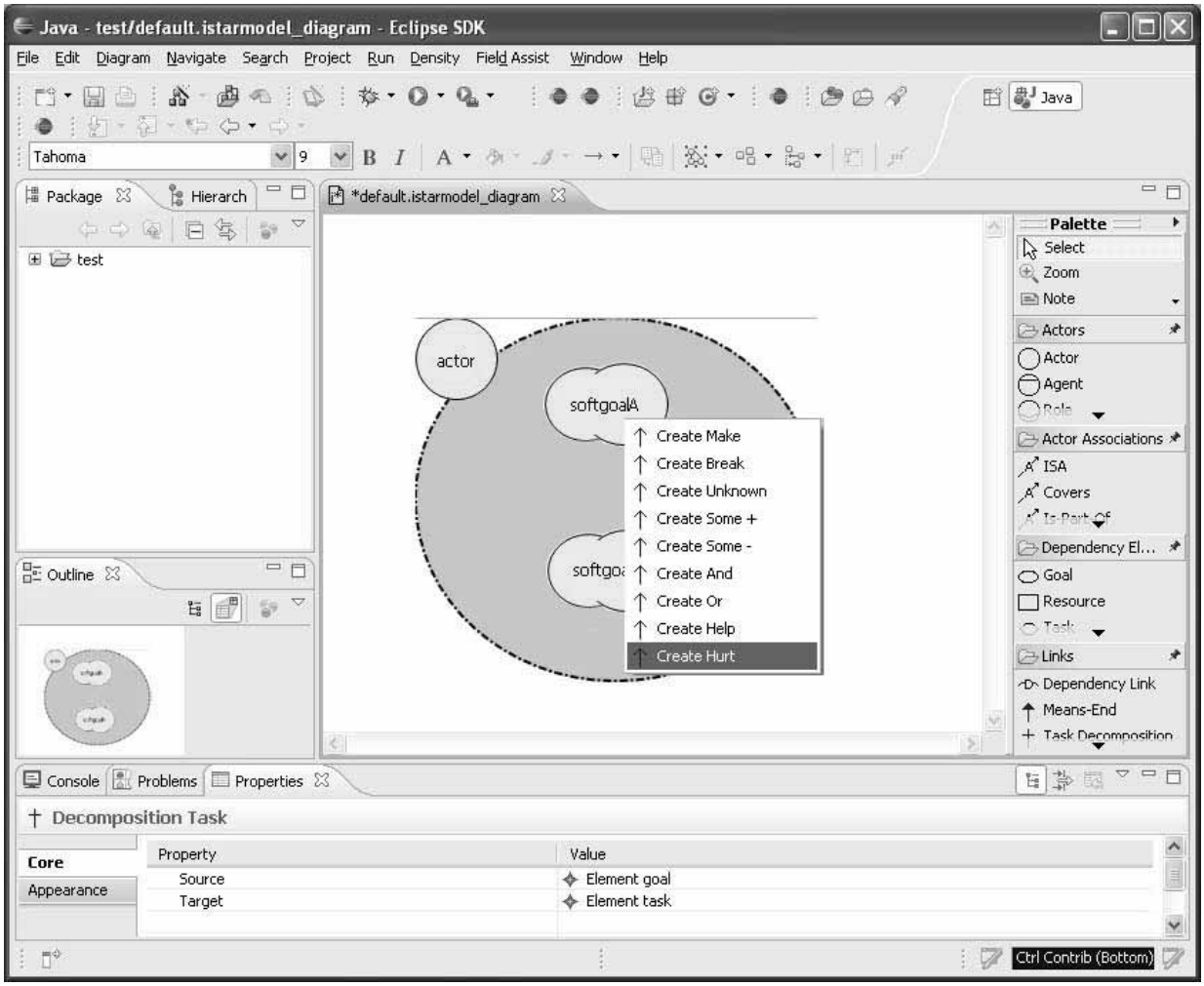


Figura 108 - Resultado de aplicação do Cenário 15 à ferramenta: menu de opções de criação de link de contribuição.

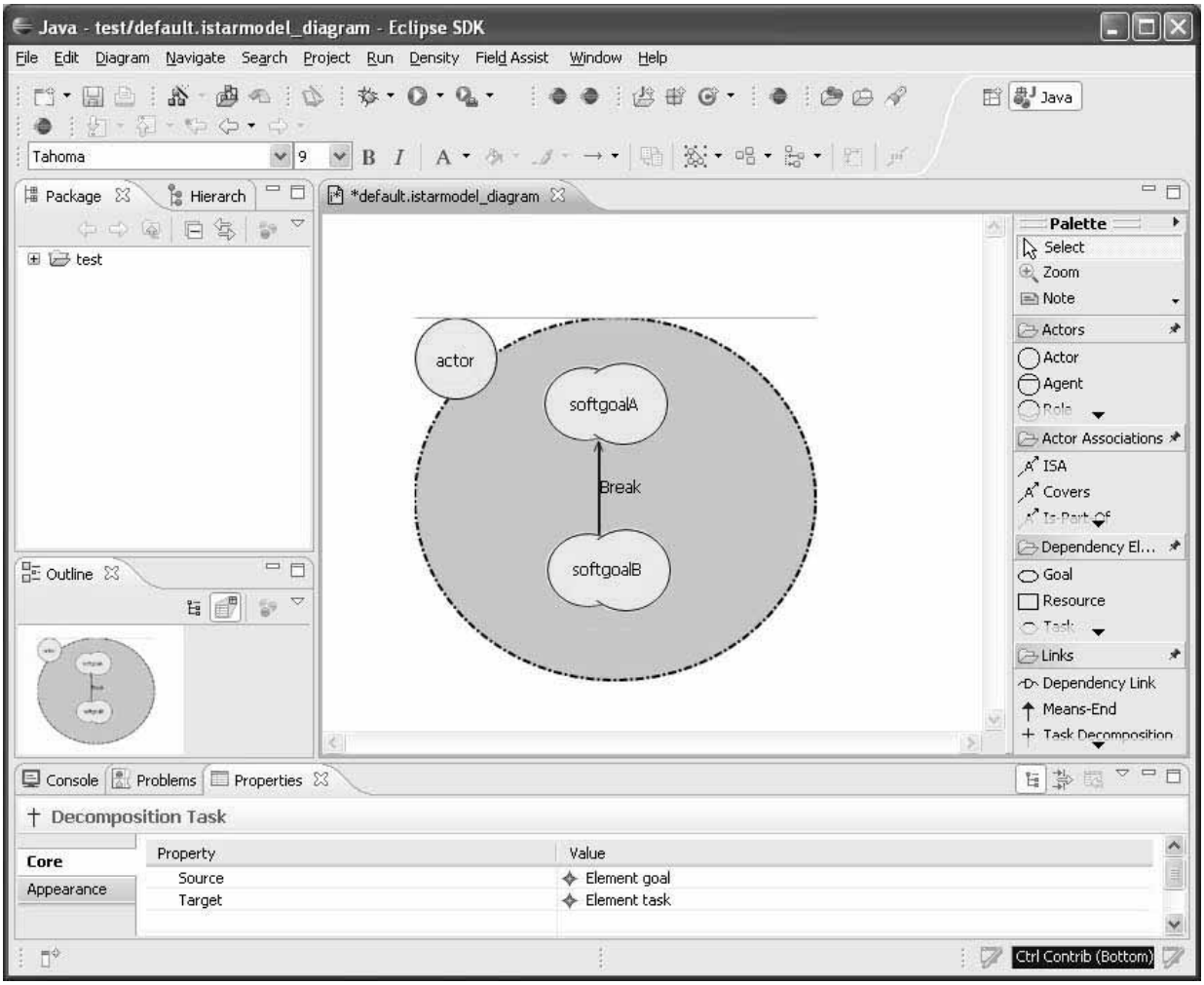


Figura 109 - Resultado de aplicação do Cenário 15 à ferramenta.

Cenário 16: *De acordo com o apresentado na Tabela XV do catálogo de [Santos 2008] - Não se deve usar Link de Contribuição fora da fronteira do ator. A ligação de contribuição é feita somente entre elementos internos à fronteira do ator.*

Utilizando a ferramenta tentamos criar um link de contribuição entre dois objetivo-*soft* fora da fronteira de um ator. A ferramenta não permite que a ligação seja realizada como pode ser visualizado na *Figura 110*.

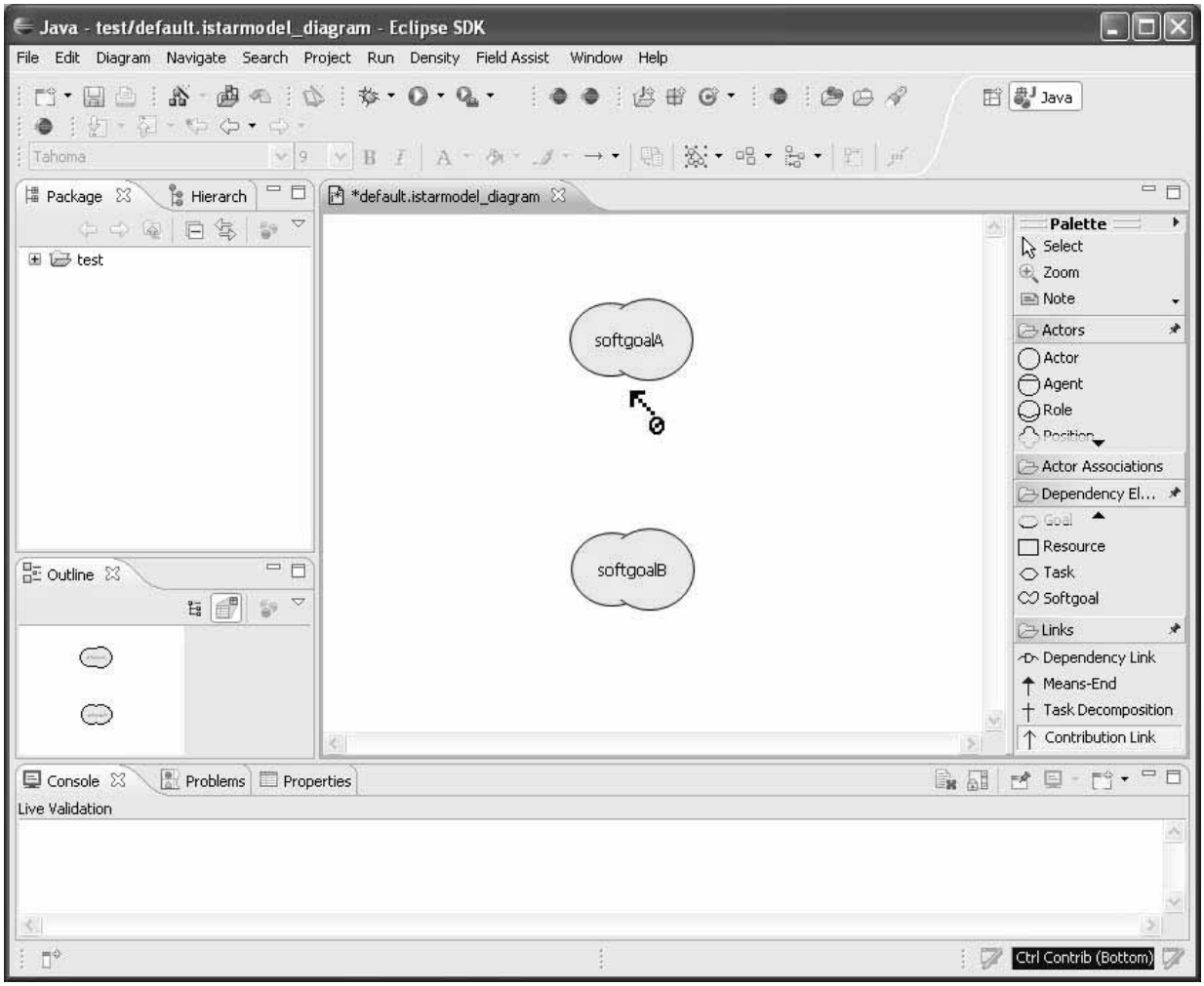


Figura 110 - Resultado de aplicação do Cenário 16 à ferramenta.

5.2 Discussão

A ferramenta IStar Tool, modelada e descrita no capítulo 4 desta dissertação segundo o *framework* GMF [GMF 2008] (baseado na criação de modelos), foi apresentada neste capítulo. O objetivo foi apresentar a utilização da ferramenta IStar Tool verificando o seu funcionamento de acordo com as regras em OCL definidas na seção 4.3.5 desta dissertação. Para tanto, foram aplicados cenários de acordo com guia do i* [Grau 2008] e com o catálogo de [Santos 2008] utilizados como referência para a construção das restrições e colocados em anexo à este trabalho, Anexo B e Anexo A, respectivamente.

Foram aplicados cenários relativos ao:

(i) Uso de Links de Associação - permitida somente entre atores.

(ii) Reuso de Dependum em uma relação de dependência.

(iii) Uso de Link de Dependência - que não é permitido dentro da fronteira de atores, tem uma representação própria através do uso do símbolo “D”, e deve ser usado sendo obrigatória a apresentação do *Dependum*.

(iv) Uso de Fronteira de Ator - na qual não é permitida a adição de outro ator.

(v) Uso de Link Meio-fim - que não pode ser criado fora da fronteira do ator, e só deve ser utilizada para ligar uma tarefa a um objetivo.

(vi) Uso do link de Decomposição de Tarefas - que não pode ser criado fora da fronteira de um ator e só deve ser utilizado para ligar elementos (objetivo, objetivo-*soft*, recurso e/ou tarefa) a uma tarefa.

(vii) Uso do Link de Contribuição - para ligar elementos (objetivo, objetivo-*soft*, recurso e/ou tarefa) a um objetivo-soft, e não pode ser criado fora da fronteira de um ator.

Na aplicação dos cenários resumidos acima a ferramenta se comportou de forma apropriada. Porém vale registrar que nem todos os itens do guia do i* [Grau 2008] e do catálogo de erros mais comuns de [Santos 2008] foram atendidos. Itens relativos às boas práticas para nomeação dos elementos e boas práticas para uso destes elementos não foram atendidos, pois estes itens tratam de questões subjetivas (de difícil verificação) dependendo, portanto da interpretação dos elementos. A ferramenta IStar Tool tem como objetivo permitir a criação de modelos válidos focando na construção apropriada dos relacionamentos.

Capítulo

6 Conclusão e Trabalhos Futuros

Este capítulo apresenta as conclusões a cerca do trabalho apresentado e os trabalhos que poderão ser desenvolvidos a partir do mesmo.

6.1 Contribuições do Trabalho

É essencial para a Engenharia de Software Orientada a Agentes ter a cobertura do processo de desenvolvimento de software. As metodologias orientadas a agente são inerentemente intencionais, o que mostra claramente a necessidade de se focar no estudo do ambiente organizacional onde o sistema em desenvolvimento irá operar. Entender este ambiente pode ajudar a reduzir consideravelmente a incompatibilidade entre o sistema e seu ambiente operacional. Assim sendo, fica clara a importância do Projeto Tropos (que tem como foco a etapa de requisitos) e da necessidade de se prover um ambiente de desenvolvimento capaz de dar suporte as atividades de cada fase de seu processo.

O Tropos utiliza como linguagem de modelagem, das fases de requisitos iniciais e finais, o i* [Yu 1995]. O *framework* i* possui uma estrutura conceitual rica, capaz de inserir no processo de desenvolvimento de software conceitos intencionais e sociais. O i* foi inicialmente proposto por [Yu 1995], mas hoje existem algumas extensões ou variações para sua versão original. Essas variações surgiram de diferentes grupos de pesquisa para atender ao propósito particular de cada um deles, e com isso surgiram diversas ferramentas de suporte. Para esta dissertação foi utilizado como referência a evolução do i* original publicada em [i* Wiki 2008][Grau 2008]. Essa evolução da descrição original do i* [Yu 1995] tem ocorrido ao longo da publicação de vários artigos [Yu 2001a][Yu 2001b][Yu 2002][Yu e Liu 2005] referenciados em [i* Wiki 2008]. O uso desta referência se justifica, pois este projeto foi criado com a intenção de permitir o trabalho colaborativo a cerca do i *, possibilitando aos usuários desta linguagem de modelagem uma visão comum de seu uso. Esse entendimento comum é importante para que todos tenham a mesma visão dos sistemas modelados utilizando o i*, não produzindo assim um sistema em desacordo com o ambiente. Além desta referência utilizamos também o catálogo dos erros mais comuns na construção de modelos em i*, publicados em [Santos 2008] para a criação da ferramenta proposta nesta dissertação.

Hoje, existem diferentes ferramentas disponíveis para modelar em i*, porém estas possibilitam a construção de modelos incoerentes (sem validação) ou não atendem ao processo do Tropos definido em [Silva, M. J. 2007] - utililzado como referência nesta dissertação.

Após o estudo detalhado da linguagem i* e das ferramentas disponíveis, apresentamos a implementação de uma ferramenta - *plug-in* para o Eclipse - para dar suporte à modelagem i* nas fases de requisitos inicial e final do Tropos [Castro 2002] de acordo com os guias e boas práticas publicadas em [i* Wiki 2008][Grau 2008][Santos 2008]. A utilização do guia do i* [i* Wiki 2008][Grau 2008] como direcionador da implementação é um ponto importante deste trabalho pois o projeto da comunidade do [i* Wiki 2008] tem como objetivo tornar a linguagem amplamente utilizada e padronizada.

Para a criação da ferramenta IStar Tool foi realizado um estudo sobre o *framework* de desenvolvimento GMF foi realizado, e sua utilização possibilitou o desenvolvimento baseado em modelos, o que tornou o processo de implementação mais abstrato. O processo se mostrou bastante simples, com exceção de alguns pontos de refinamento sobre o comportamento da ferramenta que ainda não é totalmente automatizado. Como por exemplo, a necessidade de criação de código Java

para representar a figura do link de dependência, como apresentado na seção 4.3.3 na definição do modelo gráfico.

Outro fator importante sobre a adoção de tal plataforma, é a possibilidade de definição de restrições em OCL sobre os modelos gerados pelo *framework* (seção 4.3.5), o que possibilita a verificação dos modelos criados pela ferramenta.

Dentre as vantagens de se utilizar a ferramenta proposta incluem:

(i) Utilização da versão da comunidade do i* que tem como objetivo principal gerar uma padronização do seu uso, coibindo a criação de modelos incoerentes ou inconsistentes;

(ii) Possibilidade de validação dos modelos i* criados através das regras OCL definidas;

(iii) Possibilidade de extensão do modelo definido para a ferramenta para que a mesma apóie outras atividades relativas às demais fases do processo do Tropos segundo [Silva, M. J. 2007]. Esta possibilidade se deve ao fato deste trabalho ter adotado os *frameworks* do Eclipse (GMF, GEF, EMF). Para realização da extensão é preciso que, utilizando o *framework* GMF meta-modelos, modelos gráficos, e modelos ferramentais sejam criados para cada atividade do processo e que estes sejam associados ao modelo de mapeamento definido nesta dissertação;

(iv) Possibilidade de realizar transformação entre modelos, como por exemplo: transformar modelos i* em modelos de classe;

(v) Possibilidade de integração da ferramenta com outros recursos (*plug-ins*), tanto os recursos disponibilizados pelo próprio Eclipse (como por exemplo, o projeto UML2 [Eclipse - UML2 2008] que é uma implementação do meta-modelo da UML 2.0), quanto outros recursos como as plataformas de desenvolvimento orientado a agentes tais como JADE [JADE 2007].

6.2 Trabalhos Futuros

Para trabalhos futuros temos a necessidade de extensão da ferramenta para que a mesma possa atender as demais fases do processo Tropos definido em [Silva, M. J. 2007]. Além disso, será importante a relação desta dissertação com outros trabalhos que estão sendo desenvolvidos pelo Laboratório de Engenharia de Requisitos (LER) da Universidade Federal de Pernambuco, a fim de que estes sejam validados e casos de uso sejam gerados. Como exemplos de trabalhos têm-se: a utilização de métricas para avaliar a qualidade dos modelos i* produzidos como proposto por [Santos 2008], a utilização de aspectos (*Aspectual IStar*) como uma proposta para lidar com a complexidade dos modelos i* [Alencar 2007][Alencar 2008], e a possibilidade de se realizar rastreabilidade nas fases de requisitos do Tropos construindo matrizes de rastreamento como proposto por [Pinto 2008].

Outro importante trabalho a ser desenvolvido na ferramenta será a implementação da possibilidade de importar modelos i* descritos em outras ferramentas, tais como OME [Yu e Liu 2005], OpenOME [OpenOME 2006], e TAOM4E [TAOM4E 2006]. Para isto, os meta-modelos utilizados por

outras ferramentas deverão ser estudados em detalhes para que seja gerado um mapeamento entre o que é gerado por estas e como representamos isso na ferramenta implementada neste trabalho.

A adoção da tecnologia (plataforma Eclipse e *frameworks*) também irá nos permitir a extensão da ferramenta para que outros recursos possam ser integrados a ela. Como por exemplo:

(i) A integração do projeto UML2 do Eclipse [Eclipse - UML2 2008] para permitir a geração dos artefatos de saída da fase de Projeto Detalhado do Tropos [Silva, C. T. L. L. 2007][Silva, C. 2007];

(ii) A integração da plataforma de desenvolvimento JADE [JADE 2007] para permitir a geração de código para a ferramenta como saída da fase de implementação.

Um passo importante a ser tomado é a divulgação da ferramenta para a comunidade que usa o i* segundo [i* Wiki 2008][Grau 2008] e para a comunidade que usa o Tropos segundo o processo definido por [Silva, M. J. 2007] através da construção de um portal ou Wiki através do qual as referências utilizadas neste trabalho, bem como a ferramenta implementada estarão disponíveis.

Referências Bibliográficas

[Alencar 2008] Alencar, F. M. R. ; Castro, J. F. B. ; Monteiro, C. ; Ramos, R. A. ; Santos, E. . Towards Aspectual i*. In: The 3rd International i* Worshop, 2008, Recife. iStar'08 - Workshop Proceedings, 2008. v. 322. p. 1-4.

[Alencar 2007] Alencar, F. M. R. ; Moreira, A. M. D. ; Araújo Júnior, J. B. da S. ; Castro, J. F. B. ; Ramos, R. A. ; Silva, C. T. L. L. . Proposal to deal with the Complexity of i* Models with Aspects. In: First International Conference on Research Challenges in Information Science (RCIS), 2007. Proceedinf of the First International Conference on Research Challenges in Information Science (RCIS), 2007.

[Ayala 2006] Ayala, C.; Cares, C.; Carvalho, J.; Grau, G.; Haya, M.; Salazar, G.; Franch, X.;Mayol, E.; Quer, C.: “A Comparative Analysis of i*-Based Agent-Oriented Modeling Languages”, Proceedings of SEKE, 2006.

[Bertolini 2006] Bertolini, D.; Novikau, A.; Susi, A.; Perini, A.: TAOM4E: an Eclipse ready tool for Agent-Oriented Modeling. Issue on the development process. Technical Report, ITC-irst 2006.

[Bertolini 2005] Bertolini, D., Perini, A., Susi, A., and Mouratidis, H.: The Tropos Visual Language. A MOF 1.4 Compliant Metamodel. Agentlink III AOSE TFG 2. Ljubljana (Slovenia), 2005

[Bézivin 2005] Bézivin, J.: On the unification power of models. Software and System Modeling, Vol. 4, N. 2, 2005, pp. 171-188.

[Bresciani 2004] Bresciani, P.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J.;Perini, A.: Tropos: An Agent -Oriented Software Development Methodology. In Autonomous Agents and Multi –Agent Systems v. 8 (3): 203-236, May 2004.

[Castro 2006] Castro, J.; Alencar, F.; Silva, C. T. L. L.: Engenharia de Software Orientada a Agentes. In: Karin Breitman and Ricardo Anido. (Org.). Atualizações em Informática, Rio de Janeiro: Editora PUC-Rio, 2006, v., p. 245-282.

[Castro 2005] Castro, J.; Giorgini, P.; Kethers. S.; Mylopoulos, J.: A Requirements-Driven Methodology for Agent-Oriented Software . In B. Henderson-Sellers and P. Giorgini (Eds) Agent-Oriented Methodologies, Idea Group. 2005.

[Castro 2004] Castro, J.; Mylopoulos, J.; Silva, C. T. L. L. : Agent-Driven Requirements Engineering. In: Julio Cesar S. P. Leite, Jorge H. Doorn. (Org.). Perspectives on Software Requirements. Boston: Kluwer Academic Publishers, 2004, v. 1, p. 253-274.

[Castro 2002] Castro, J., Kolp, M., Mylopoulos, J.: Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems Journal, Elsevier, Vol. 27: 365-389, 2002.

[Castro 2001] Castro, J., Kolp, M., Mylopoulos, J.: UML for Agent-Oriented Software Development: the Tropos Proposal. In: International Conference on the Unified Modeling Language, 2001, Toronto. Proceedings of the 4th International Conference on the Unified Modeling Language. London: Springer Verlag -, 2001. v. 1. p. 414-433.

[Chung 2000] Chung, L. K.; Nixon, B.; Yu, E.; Mylopoulos, J.: Non-Functional Requirements in Software Engineering, Kluwer Publishing, 2000.

[Cimatti 2002] Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.. NUSMV 2: An opensource tool for symbolic model checking. In Computer Aided Verification, number 2404 in LNCS, Copenhagen (DK), July 2002. Springer.

[DeLoach 2006] DeLoach, S.: Engineering Organization-based Multiagent Systems. In: Proceedings of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05), Saint Louis, EUA, ACM Press, p. 1– 7, 2006.

[DesCARTES Architect 2007] DesCARTES Architect, Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems. Disponível em: http://www.isys.ucl.ac.be/descartes/index.php, 2007. Último acesso em: 22 de Dezembro de 2007.

[Do 2004] Do, T. T.; Kolp M. e Faulkner, S.: Agent Oriented Design Patterns: The SKwyRL Perspective. In: Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS'04), Porto, Portugal, p. 48 – 53, 2004.

[Eclipse 2008] Eclipse, The Eclipse Foundation. Disponível em: http://www.eclipse.org/, 2008. Último acesso em: 27 de Junho de 2008.

[Eclipsepedia 2008] Eclipsepedia - Eclipse Wiki. Disponível em: http://wiki.eclipse.org/index.php/, 2008. Último acesso em: 27 de Junho de 2008.

[Eclipse - UML2 2008] Eclipse Project UML2. Disponível em: http://www.eclipse.org/modeling/mdt/?project=uml2, 2008. Último acesso em: 27 de Junho de 2008.

[EMF 2008] EMF - Eclipse Modeling Framework. Disponível em: http://www.eclipse.org/emf, 2008. Último acesso em: 27 de Junho de 2008.

[Ferber e Gutknecht 2000] Ferber, J.; Gutknecht, O.: Admission of agents in groups as a normative and organizational problem, In: Workshop on Norms and Institutions in Multi-agent systems at Normative Agent, ACM press, 2000.

[FIPA 2007] FIPA (The Foundation for intelligent agents), Disponível em: http://www.fipa.org, 2007. Último acesso em: 14 de Maio de 2007.

[FIPA-OS 2007] FIPA-OS: A component-based toolkit enabling rapid development of FIPA compliant agents. Disponível em: http://sourceforge.net/projects/fipa-os/, 2007. Último acesso em: 14 de Maio de 2007.

[Gans 2004] Gans , G., Schmitz, D., Arzdorf, T., Jarke, M., Lakemeyer, G.. SNet reloaded: Roles, monitoring, and agent evolution. In AOIS, LNCS 3508. Springer, 2004.

[GEF 2008] GEF - Graphical Editing Framework. Disponível em: http://www.eclipse.org/gef/, 2008. Último acesso em: 27 de Junho de 2008.

[Giorgini 2008] Giorgini, P.; Mylopoulos, J.; Penserini, L.; Perini, A.; Susi, A.: Tropos at the Age of Eight : On-going Research at FBK, UniTN and UT. In Proceedings of the 3rd International I* Workshop, ISSN 1613-0073, pages 83-90, 2008.

[Giorgini 2005a] Giorgini, P.; Kolp, M.; Mylopoulos, J.; Castro, J.: Tropos: A Requirements-Driven Methodology for Agent-Oriented Software. In: Paolo Giorgini; Brian Henderson-Sellers. (Org.). Agent-Oriented Methodologies. 1 ed. Hershey, PA 17033, USA: Idea Group, Inc., 2005, v. , p. 20-45.

[Giorgini 2005b] Giorgini, P. ; Massacci F. ; Mylopoulos J. ; Siena A. ; Zannone N. : ST-Tool: A CASE Tool for Modeling and Analyzing Trust Requirements. In Proceedings of the Third International Conference on Trust Management (iTrust 2005), LNCS 3477, pages 415-419. Springer-Verlag GmbH, 2005. Disponível em: http://sesa.dit.unitn.it/sistar_tool/e107_files/public/papers/gior-mass-mylo-sien-zann-05-iTrust.pdf

[GMF 2008] GMF - *Graphical Modelling Framework*. Disponível em: http://www.eclipse.org/modeling/gmf/, 2008. Último acesso em: 27 de Junho de 2008.

[GR-Tool 2004] GR-Tool, Goal Reasoning Tool. Disponível em: http://Troposproject.org/tools/grtool/, 2004. Último acesso em: 22 de Dezembro de 2007.

[Grau 2005a] Grau, G.; Franch, X.; Maiden, N. A. M.: REDEPEND-REACT: an Architecture Analysis Tool. In: Proceedings of the 13th IEEE Requirements Engineering International Conference (RE'05). August 29 - September 2, 2005. Paris, France. Pages: 455 - 456. Disponível em: http://www.lsi.upc.edu/~ggrau/publications/ggrau-Redepend-React-RE05.pdf

[Grau 2005b] GRAU, G. et al. Risd: A methodology for building i* strategic dependency models. In: Proceedings of The Seventeenth International Conference on Software Engineering and Knowledge Engineering, SEKE 2005. [S.l.: s.n.], 2005. p. 259-266.

[Grau 2006] Grau, G., Franch, X., Ávila, S.. J-PRiM: A Java Tool for a Process Reengineering i* Methodology. In Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06). 11-15 September 2006. Minneapolis/St Paul, Minnesota, United States. Pages: 352-353.

[Grau 2008] Grau, G., Yu, E.; Horkoff, J., Abdulhadi, S.: i* Guide. Disponível em: http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=67, 2008. Último acesso em: 16 de Julho de 2008.

[Horkoff 2006] Horkoff, J. M. Using i* Models for Evaluation. Dissertação (Mestrado) - Department of Computer Sciences, University of Toronto, 2006.

[i* Wiki 2008] I* Wiki, Disponível em: http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=2, 2008. Último acesso em: 16 de Julho de 2008.

[IBM RSA 2007] IBM RSA - *IBM Rational Software Architect*. Disponível em: http://www-306.ibm.com/software/awdtools/architect/swarchitect/, 2007. Último acesso em: 27 de Junho de 2008.

[IEEE 2000] IEEE Computer Society. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems: IEEE Std 1471. 2000.

[JACK 2007] JACK Intelligent Agents. Disponível em: http://www.agent-software.com/, 2007. Último acesso em: 14 de Maio de 2007.

[JADE 2007] JADE - *Java Agent DEvelopment Framework.* Disponível em: http://jade.cselt.it/, 2007. Último acesso em: 14 de Maio de 2007.

[JADEX 2007] JADEX - *BDI Agent System*. Disponível em: http://vsis-www.informatik.uni-hamburg.de/projects/jadex/features.php, 2007. Último acesso em: 14 de Maio de 2007.

[JAVA 2008] JAVA, Java Technology, Disponível em: http://www.java.sun.com/, 2008. Último acesso em: 27 de Junho de 2008.

[Jennings 2000a] Jennings, N. R.; Norman, T. J.; Faratin, P.; O'Brien, P.; Odgers, B.: Autonomous agents for business process management. Applied Articial Intelligence Journal, 14(2):145-190, 2000.

[Jennings 2000b] Jennings, N. R.; Norman, T. J.; Faratin, P.; O'Brien, P.; Odgers, B.: Implementing a business process management system using ADEPT: A real-world case study. Applied Articial Intelligence Journal, 14(5):421-490, 2000.

[Jennings 2000c] Jennings, N.: On Agent-Based Software Engineering. In Bradshaw, J. (ed): Handbook of Agent Technology, AAAI/MIT Press, 2000.

[Jennings e Wooldridge 2001] Jennings, N.; Wooldridge, M.: Agent-oriented software engineering. In Bradshaw, J. (Ed.), Handbook of agent technology. AAAI/MIT Press, 2001.

[J-PRiM 2006] J-PRim, a Java tool for a Process Reengineering i* Methodology. Disponível em: http://www.lsi.upc.edu/~ggrau/JPRIM/, 2006. Último acesso em: 22 de Dezembro de 2007.

[Kolp 2001] Kolp, M.; Castro, J.; Mylopoulos, J.: A social organization perspective on software architectures. In: Proceedings of the 1st International Workshop from Software Requirements to Architectures (STRAW'01), Toronto, Canada, p. 5 – 12, 2001.

[Kolp 2002] Kolp, M.; Giorgini, P.; e Mylopoulos, J.: Information Systems Development through Social Structures. In: Proceedings 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, ACM Press, p. 183 – 190, 2002.

[Kolp 2005] Kolp, M.; Do, T.; Faulkner, S. e Hoang, H.: Introspecting Agent Oriented Design Patterns. In: Handbook of Software Engineering and Knowledge Engineering, Vol. 3: Recent Advances, Edited by Chang, S. K. World Scientific, p. 151-176, 2005.

[Kolp 2006] Kolp, M.; Giorgini, P.; and Mylopoulos, J.: "Multi-Agents Architectures as Organizational Structures". In Journal of Autonomous Agents and Multi-Agent (JAAMAS), 13(1):3-25, Springer, 2006. Disponível em: http://www.troposproject.org/papers_files/1org-mas-ijcis.pdf.

[Kotonya e Sommerville 1998] Kotonya, G. and Sommerville, I.: Requirements Engineering – Processes and Techniques. John Willy & Sons, ISBN: 0-471-97208-8, 1998.

[Lucena 2008] Lucena, M. J. N. R. ; Santos, E. B. ; Silva, M. J. ; Silva, C. T. L. L. ; Alencar, F. M. R. ; Castro, J. F. B. : Towards a Unified Metamodel for i*. In: IEEE International Conference on Research Challenges in Information Science RCIS'08, 2008, Marrakech. Proceedings of the IEEE International Conference on Research Challenges in Information Science - RCIS'08, 2008, v. 1. p. 1-10.

[Luck 2003] Luck, M., McBurney, P., Preist, C.: Agent Technology: Enabling Next Generation Computing. AgentLink (2003), ISBN 0854 327886., 94 pages. Disponível em: http:// www.agentlink.org/roadmap.

[MDA 2008] MDA - OMG Model Driven Architecture. Disponível em: http://www.omg.org/mda/, 2008. Último acesso em: 27 de Junho de 2008.

[Mylopoulos e Castro 2000] Mylopoulos, J. ; Castro, J. . Tropos: A Framework for Requirements-Driven Software Development. In: Sjaak Brinkkemper; Eva Lindercrona; Arne Solvberg, A (eds). (Org.). Information Systems Engineering; State of the Art and Research Themes. 1 ed. London: Springer-Verlag London Berling Heidelberg, 2000, v. 1, p. 261-273.

[OCL 2007] OCL - Object Constraint Language Specification. Disponível em: http://www.omg.org/technology/documents/formal/ocl.htm, 2007. Último acesso em: 27 de Junho de 2008.

[Odell 2000] Odell, J.; Parunak, H.; Bauer, B.: Extending UML for Agents. In: Wagner, G.; Lesperance, Y., Yu, E. (Eds.), Proceedings of the Agent-Oriented Information Systems, Workshop at the 17th National Conference on Artificial Intelligence, 2000.

[Odell 2001] Odell, J.; Parunak, H.V. D.; Bauer, B: Representing Agent Interaction Protocols in UML. Agent-Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds. (121-140), Springer-Verlag, Berlin, (Held at the 22nd International Conference on Software Engineering (ISCE)), 2001.

[OMG 2008] OMG, The Object Management Group. Disponível em: http://www.omg.org/, 2008. Último acesso em: 27 de Junho de 2008.

[OpenOME 2006] OpenOME, Disponível em: https://se.cs.toronto.edu/trac/ome/wiki (Home Page do Projeto), 2006. Último acesso em: 22 de Dezembro de 2007.

[Perini e Susi 2005] Perini, A.; Susi, A.: Automating Model Transformations in Agent-Oriented Modelling. Proceedings of 6th International Workshop AOSE 2005, Utrecht, NL, Julho 25-26, 2005.

[Penserini 2006] Penserini, L. ; Perini, A. ; Susi, A. ; Mylopoulos, J.: From Stakeholder Intentions to Software Agent Implementations. Proceedings of the 18th Conference on Advanced Information Systems Engineering (CAiSE-06), Springer Verlag, LNCS-4001, Luxembourg, Junho, 2006.

[Pinto 2008] Pinto, R. C. C.. Improving Traceability in Agent Oriented Development. 2008. Tese (Doutorado em Ciências da Computação) - Universidade Federal de Pernambuco. Disponível em: www.cin.ufpe.br/~ler

[Protégé 2007] Protégé, *Stanford Center for Biomedical Informatics Research*. Disponível em: http://protege.stanford.edu/, 2007. Último acesso em: 22 de Dezembro de 2007.

[Rao e Georgeff 1995] Rao, A.; Georgeff, M.: BDI agents: from theory to practice. In: Technical Note 56, Australian Artificial Intelligence Institute, 1995.

[Redepend-React 2006] Redepen-React, home page do projeto. Disponível em: http://www.lsi.upc.edu/~ggrau/REDEPEND-REACT/, 2006. Último acesso em: 22 de Dezembro de 2007.

[Rudowsky 2004] Rudowsky, I.: Intelligent Agents Tutorial. In: # AMCIS 2004 - Americas Conference on Information Systems, New York, NY, Aug 6-8, 2004. Disponível em: http://userhome.brooklyn.cuny.edu/irudowsky/Papers/IntelligentAgentsTutorial.pdf

[Sannicoló 2002] Sannicoló F., Perini A., Giunchiglia F.: The Tropos modeling language. A User Guide. Technical report DIT-02-0061, University of Trento, February 2002.

[Santos 2008] Santos, E. B.. Uma Proposta de Métricas para Avaliar Modelos i*, 2008. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Pernambuco. Disponível em: www.cin.ufpe.br/~ler

[Shaw e Garland 1993] Shaw, M. e Garlan, D.: An Introduction to Software Architecture. In: Advances in Software Engineering and Knowledge Engineering, Vol. I, Edited by Ambriola, V. and Tortora, G., World Scientific Publishing Company, New Jersey, 1993.

[Schwambach 2004] Schwambach, M.; Pezzin, J.; Falbo, R.: OplA: Uma Metodologia para o Desenvolvimento de Sistemas Baseados em Agentes e Objetos. JIISIC'04, 4ª Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento, Novembro, 2004. Disponível em: http://www.inf.ufes.br/~falbo/download/pub/2004-JIISIC-3.pdf.

[Si* Tool 2006] Si* Tool, Security and Dependability Tropos Tool. Disponível em: http://sesa.dit.unitn.it/sttool/home.php?7, 2006. Último acesso em: 22 de Dezembro de 2007.

[Silva, C. 2003] Silva, C. T. L. L.: Detalhando o projeto arquitetural no desenvolvimento de software orientado a agentes: O caso Tropos. Dissertação de

Mestrado. Centro de Informática, Universidade Federal de Pernambuco, Fevereiro, 2003.

[Silva, C. 2006] Silva, C., Castro, J., Tedesco, P., Araújo, J., Moreira, A., Mylopoulos, J. (2006) “Improving the Architectural Design of Multi-Agent Systems: The Tropos Case”, In: 5th Software Engineering for Large-Scale Multi-Agent Systems (SELMAS’06) at ICSE’06, Shangai, China, p. 107 – 113.

[Silva, C. T. L. L. 2007] Silva, C. T. L. L.: Separating Crosscutting Concerns in Agent Oriented Detailed Design: The Social Patterns Case. 2007. Tese (Doutorado em Ciências da Computação) - Universidade Federal de Pernambuco, Conselho Nacional de Desenvolvimento Científico e Tecnológico.

[Silva, C. 2007] Silva, C. ; Araújo, J. ; Moreira, A. ; Castro, J. F. B. . Designing Social Patterns using Advanced Separation of Concerns. In: 19th International Conference Advanced Information Systems Engineering (CAiSE 07), 2007, Trondheim. Proceedings of CAiSE'07 (Qualis A Internacional), 2007.

[Silva, I. 2005] Silva, I. G. L. Projeto e Implementação de Sistemas Multi-Agentes: O Caso Tropos, Universidade Federal de Pernambuco, 2005. Dissertação de mestrado.

[Silva, J. M. C. da 2006] Silva, J. M. C. da; Silveira, R. A. Modelagem de Objetos Inteligentes de Aprendizagem utilizando a metodologia MaSE. RENOTE. Revista Novas Tecnologias na Educação, v. 1, p. 1, 2006.

[Silva, M. J. 2007] Silva, M. J.; Maciel P.R. M.; Pinto, R. C.; Alencar, F.R.; Tedesco P.; Castro, J.: Extracting the Best Features of Two Tropos Approaches for the Efficient Design of MAS, In Workshop ibero-americano de Engenharia de Requisitos e ambientes de Software - IDEAS’2007, 2007. Isla de Margarita - Venezuela. Proceedings of the 10th Iberoamerican Workshop on Requirements Engineering and Software Environments, 2007. p. 1-10.

[Susi 2005] Susi , A.; Perini A.; Giorgini P.; Mylopoulos J.: The Tropos Metamodel and its Use. Informatica, 29(4):401--408, 2005.

[TAOM4E 2006] TAOM4E, Tool for Agent-Oriented Modeling for Eclipse Platform. Disponível em: http://sra.itc.it/tools/taom4e/, 2006. Último acesso em: 22 de Dezembro de 2007.

[Tropos Project 2008] Tropos Project, Requirements-Driven Development for Agent Software. Disponível em: http://www.troposproject.org/, 2008.

[Tutorial OCL 2005] Tutorial: OCL Validation Constraints, 2005. Disponível em: http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.validation.doc/tutorials/oclValidationTutorial.html. Último acesso em: 12/10/2008.

[T-Tool 2003] T-Tool, The Formal Tropos Tool. Disponível em: http://dit.unitn.it/~ft/ft_tool.html, 2003. Último acesso em: 22 de Dezembro de 2007.

[Venkatesan 2006] Venkatesan, M. D.: Generation of Diagram Editors, Taking the Enterprise Application Integration Patterns as Case Study. Master Thesis. Hamburg, Alemanha. 2006.

[Wautelet 2005] Wautelet, Y.; Kolp, M.; e Achbany, Y.: S-Tropos, An Interative SPEM-Centric Software Project Management Process, Working Paper IAG, 2005.

[Webster 2005] Webster, I.; Amaral, J.; Cysneiros, L. M. A survey of good practices and misuses for modelling with i*. In: Proceedings of VIII Workshop on Requirements Engineering (WER 2005). Porto, Portugal: [s.n.], 2005.

[Weiss 2002] Weiss, G.: Agent Orientation in Software Engineering. Knowledge Engineering Review, Vol. 16, n. 4, (2002): 349-373.

[Wooldridge 2002] Wooldridge, M.: Introduction to Multi-Agent Systems. Jonh Wiley and Sons, New York (2002).

[Wooldridge 2000] Wooldridge, M.; Jennings, N. R.; Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems, 3 (3):285-312 (2000).

[Wooldridge 1997] Wooldridge, M.: Agent-based software engineering. IEE Proc. On Software Engineering, 144 (1) 26-37, 1997.

[Wooldridge e Ciancarini 1999] Wooldridge, M.; Ciancarini, P.: Agent-Oriented Software Engineering. Handbook of Software Engineering and Knowledge Engineering Vol. 0, No. 0, 1999.

[XMI 2007] XMI - MOF 2.0 / *XMI Mapping Specification*, v2.1.1. Disponível em: http://www.omg.org/technology/documents/formal/xmi.htm, 2007. Último acesso em: 27 de Junho de 2008.

[Yu e Liu 2005] Yu, E.; Liu, L.: OME (Organization Modeling Environment), Disponível em: http://www.cs.toronto.edu/km/ome (Home Page do Projeto), 2005. Último acesso em: 22 de Dezembro de 2007.

[Yu 2002] Yu, E.: Agent-Oriented Modelling: Software Versus World, In: Proceedings of the Agent-Oriented Software Engineering (AOSE'01), Edited by Wooldridge, M., Weiss, G. and Ciancarini, P., LNAI, Vol. 2222, Springer-Verlag, p. 206 – 225, (2002).

[Yu 2001a] Yu, E.: Agent Orientation as a Modelling Paradigm, Wirtschaftsinformatik, 43(2), 2001. Páginas 123-132. Disponível em: ftp://ftp.cs.utoronto.ca/pub/eric/WIj.pdf

[Yu 2001b] Yu, E.; Liu, L.; Li, Y.:Modelling Strategic Actor Relationships to Support Intellectual Property Management. 20th International Conference on Conceptual Modeling (ER-2001) - Yokohama, Japan, 2001.

[Yu 1995] Yu, E.: Modelling Strategic Relationships for Business Process Reengineering, Ph.D. thesis. Dept. of Computer Science, University of Toronto (1995).

[Zambonelli 2005] Zambonelli, F.: Multiagent System as Computational Organizations: the Gaia Methodology. Agent-Oriented Methodologies: Brian Henderson-Sellers, Paolo Giorgini; University of Technology, Sydney, Australia; University of Trento, Italy, 2005.

[Zambonelli 2003] Zambonelli, F.; Jennings, N. R.; Wooldridge, M. Developing Multiagent Systems: the Gaia Methodology. ACM Trans on Software Engineering and Methodology, 12(3): 317-370, 2003.

Apêndice A - Comparação entre Meta-modelos do i*

Este apêndice apresenta uma comparação entre o meta-modelo criado para a ferramenta proposta nesta dissertação - IStar Tool - e os meta-modelos publicados em [Ayala 2006][Lucena 2008].

Como apresentado na seção 3.1.1 desta dissertação, o *framework* i* foi inicialmente proposto por [Yu 1995], mas hoje existem algumas variações para sua versão original. Tais variações surgiram de diferentes grupos de pesquisa para atender ao propósito particular de cada um deles. Este trabalho, como apresentado ao longo dos capítulos toma como referência o trabalho da comunidade do i* (i* Wiki) [i* Wiki 2008]. O uso desta referência se justifica, pois este projeto foi criado com a intenção de permitir o trabalho colaborativo a cerca do i*, possibilitando aos usuários desta linguagem de modelagem uma visão comum de seu uso.

Este anexo apresenta, portanto as variantes mais representativas encontradas na literatura e faz uma comparação entre essas variações do i* com o meta-modelo - baseado em [Grau 2008][i* Wiki 2008] - criado nesta dissertação para o desenvolvimento da ferramenta IStar Tool.

Em particular, vamos focar em duas variantes do i*, que consideramos mais representativas na comunidade que utiliza o i* como linguagem de modelagem nas fases de requisitos iniciais e finais do Tropos: o i* original proposto por [Yu 1995] e o i* adaptado pelo grupo de pesquisa da Universidade de Trento - que trabalha com o Tropos versão da Itália ou Tropos'04 [Bresciani 2004][Susi 2005][Bertolini 2005].

Para comparar o meta-modelo criado para a ferramenta IStar Tool e essas duas variantes do i*, estudamos os modelos publicados nos trabalhos de [Ayala 2006] e [Lucena 2008]. A comparação é feita baseada no estudo de meta-modelo, pois de acordo com [Bézivin 2005], um meta-modelo é útil para:

(i) Explicar as características pertinentes a uma linguagem de modelagem,

(ii) Realizar comparação entre diversas variantes de uma mesma linguagem de modelagem, e

(iii) Identificar, entre as variantes de uma mesma linguagem, os pontos onde ocorrem as variações sintáticas.

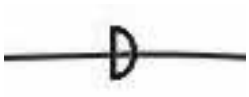
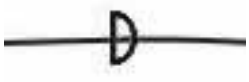
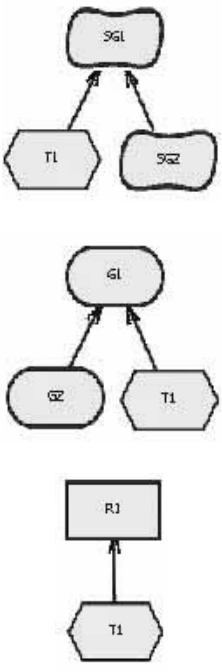
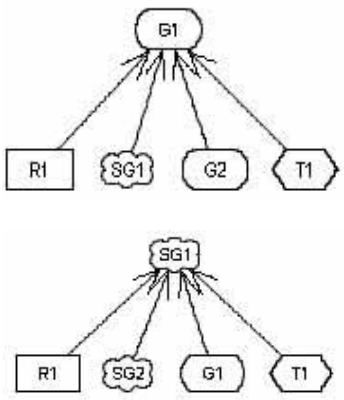
Analisando os modelos constatamos que, tanto as variações do i* quanto a referência utilizada nesta dissertação usam um mesmo conjunto de elementos estruturais para a criação dos modelos em i* - apresentados na *Tabela 21*.

Tabela 21 - Elementos Estruturais da Linguagem i*.

Elementos Estruturais da Linguagem i*	
Ator	Uma entidade que realiza ações para atingir seus objetivos. Essa entidade pode ser detalhada e ser modelada como um Agente, um Papel ou uma Posição.
Dependência	Relação de dependência entre atores. Um ator depende (*depender*) de outro (*dependee*) para algo (*dependum*).
Elemento de Dependência	O elemento (*dependum*) em torno do qual uma relação de dependência está centrada. Um elemento de dependência pode ser um objetivo, um objetivo-soft, um recurso ou uma tarefa.

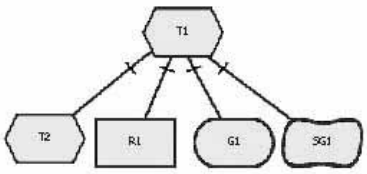
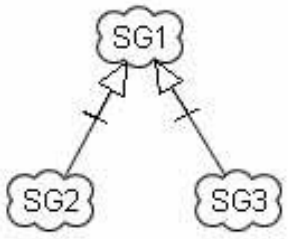
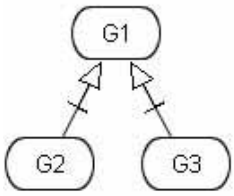
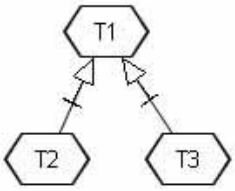
As diferenças entre as variações do i* e a referência utilizada nesta dissertação estão centradas na representação (simbologia) utilizada para representar os elementos e ligações no modelo, e na utilização dos links meio-fim, de decomposição de tarefa, e de contribuição - vide *Tabela 22*.

Tabela 22 - Diferença entre variantes do i* e o projeto [i* Wiki 2008]

i* Wiki	i* original	i* versão da Itália
Referência		
[i* Wiki 2008][Grau 2008]	[Yu 1995]	[Bresciani 2004][Susi 2005][Bertolini 2005]
Representa-ção de Link de Dependência		
		
Desta comparação o que podemos concluir é que no caso do i* versão Itália o uso desta representação pode causar confusão quanto ao entendimento correto do link. Esta representação pode ser confundida com a representação do link de contribuição.		
Links Meio-fim		
O i* Wiki afirma - guideline 5.2.1.4 - que a ligação meio-fim deve ocorrer apenas de um elemento tarefa para um objetivo. 	A tese de Yu permite as seguintes ligações meio-fim: 	O i* versão Itália permite as seguintes ligações meio-fim: 
Desta comparação o que podemos concluir é que tanto no i* original quanto no i* versão Itália o link		

meio-fim está utilizando um objetivo-*soft* como representação de um fim na relação. Enquanto que no i* Wiki para representar a um relacionamento de satisfação de um objetivo-*soft* deve-se utilizar o link de contribuição. Outra observação é que tanto no i* original quanto no i* versão Itália, qualquer elemento (objetivo, recurso, tarefa, e/ou objetivo-*soft*) é utilizado para representar um meio, em uma relação meio-fim, enquanto que segundo o i* Wiki indica como correto usar apenas o elemento tarefa para representar um meio (guidelines 5.2.1.4, 5.2.1.5 e 5.2.1.7).

Link de Decomposição

O i* wiki afirma que a ligação de decomposição deve ser da seguinte forma:	A tese de Yu permite os seguintes links de decomposição:	O i* versão Itália permite os seguintes links de decomposição:
		  

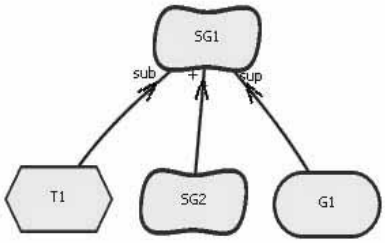
Desta comparação o que podemos concluir é que tanto no i* original quanto o i* wiki utilizam a mesma forma de representação de um link de decomposição, e ambas são utilizadas para decompor uma tarefa em outros elementos (tarefa, objetivo, recurso, e/ou objetivo-*soft*). Enquanto que no i* versão Itália é utilizada uma outra forma de representação do link e, além disso, o link é utilizado para decompor objetivos-*soft* em outros objetivos-*soft*, objetivos em outros objetivos e tarefas em outras tarefas.

Segundo a referência utilizada nesta dissertação [i* Wiki 2008][Grau 2008] objetivos-soft só podem ser decompostos utilizando-se links de contribuição (guideline 5.5.10), e para indicar a forma de atingir um objetivo deve-se utilizar a ligação meio-fim (guidelines 5.2.1.4, 5.2.1.5 e 5.2.1.7).

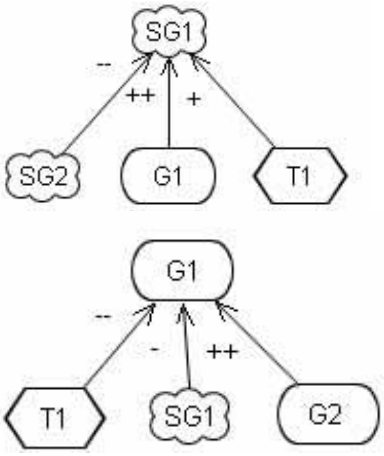
Link de Contribuição

O i* wiki afirma que a ligação de contribuição só deve ligar elementos (objetivos, recurso, tarefas, e/ou objetivos-soft) à objetivos-soft de forma que estes representem a sua contribuição para que o objetivo-*soft* seja satisfeito (guideline 5.5.10). Os tipos de contribuição são: Make, Some+, Help, Break, Some-, Hurt, Unknown, And, e OR.

A tese de Yu afirma que um link de contribuição é um link meio-fim relacionado a um objetivo-*soft* e com os rótulos de contribuição: +, -, sup (suficiente), sub (não suficiente).



O i* versão Itálila entende o link de contribuição como um outro link e que utiliza os rótulos ++ e -- para indicar as contribuições. Nesta versão do i*, o link de contribuição pode ser utilizado tanto para representar objetivos-*soft* quanto objetivos.



Desta comparação o que podemos concluir é que tanto as variações do i* (i* original e i* versão Itália) quanto o i* do wiki utilizam rótulos de contribuição diferentes. Sendo que no caso do i* do wiki ele possibilita a representação de variados níveis de contribuição. E tanto o i* wiki quanto o i* original entendem que link de contribuição deve ser utilizado para representar apenas como um objetivo-soft pode ser satisfeito, e não estende seu uso para representar também a satisfação de um objetivo como o i* versão Itália permite.

Anexo A - Catálogo dos Erros mais Freqüentes com i*

Este anexo apresenta um catálogo com os erros mais freqüentes com i.*

Tabela I - Ator sem Ligação com outros atores.

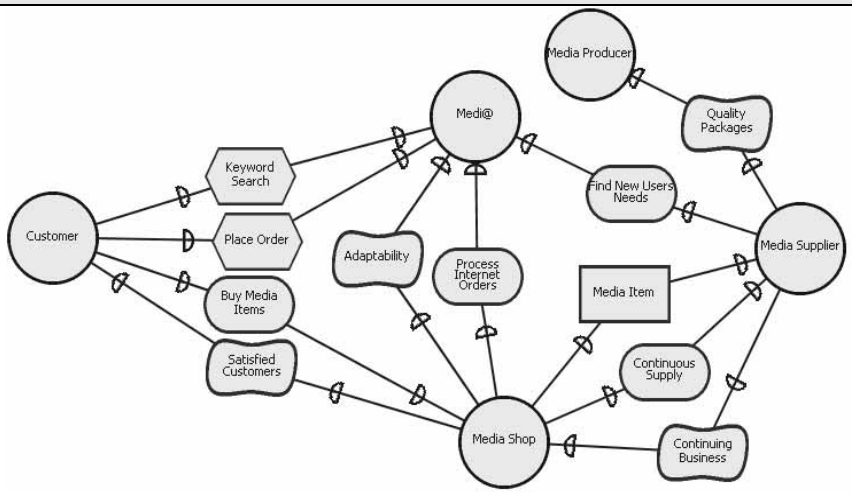
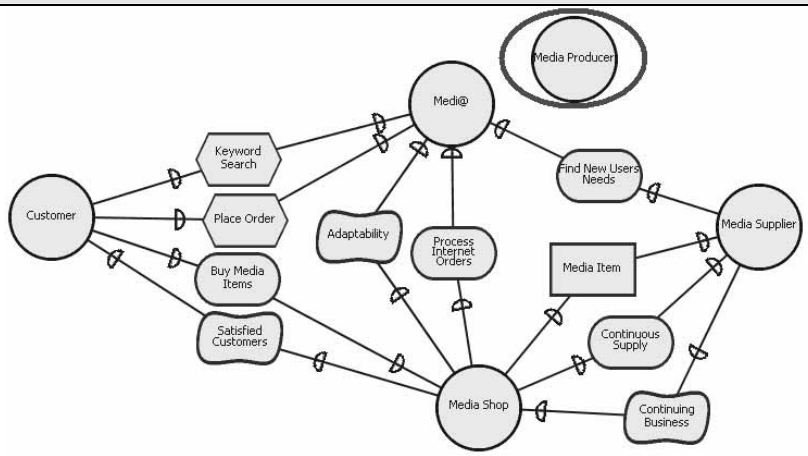
Entrada	01
Descrição	Ator sem ligação com outros atores
Observação	Um ator sem ligação com outros atores não contribui com informação para o modelo
Referência	[Webster 2005]
Questões	Existe algum ator sem nenhum tipo de ligação de dependência e sem ligações de relacionamento com outros atores (i.e., *Is-A*, *Is-part-of*, *Covers*, *Occupies*, *Plays*)?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela II - Decomposição de Goals.

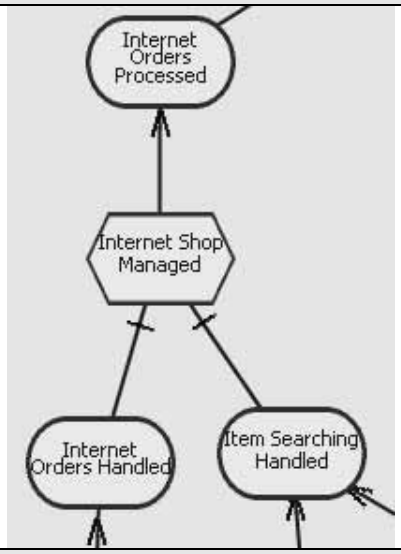
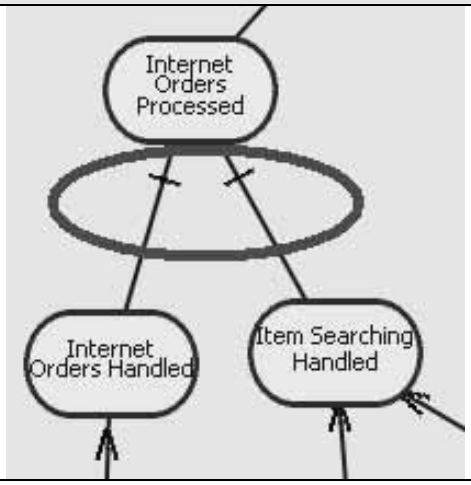
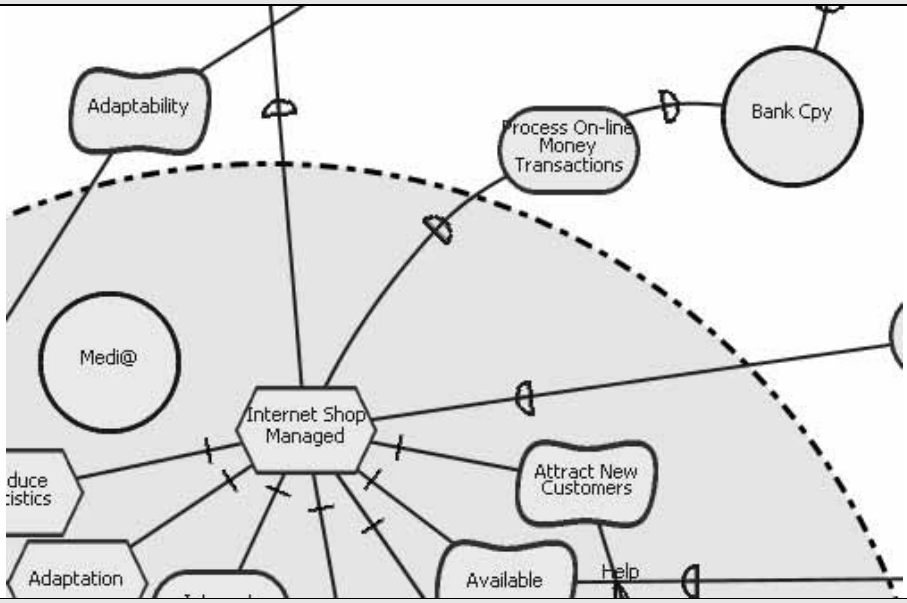
Entrada	02
Descrição	Decomposição de *Goals* em sub-*Goals*
Observação	Os *Goals* não podem se decompor em outros *Goals*, em modelos SR nenhum dos relacionamentos entre elementos intencionais está relacionados aos *Goals* nas duas extremidades de uma ligação (Decomposição, Contribuição, ou Means-Ends)
Referência	[Webster 2005][Grau 2005b][Grau 2008]
Questões	Os *Goals* do modelo SR estão ligados à outros *Goals*?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela III - Representar atores dentro das fronteiras de outros atores.

Entrada	03
Descrição	Representar atores dentro das fronteiras de outros atores
Observação	As fronteiras dos atores representam os limites da individualidade dos atores, eles não compartilham esse espaço com os atores. Apenas elementos intencionais (*Goal*, *Softgoal*, Tarefa ou Recurso)
Referência	[Webster 2005]
Questões	Existe mais de um ator em uma mesma fronteira?

Exemplo de uso Correto



Exemplo de uso Incorreto

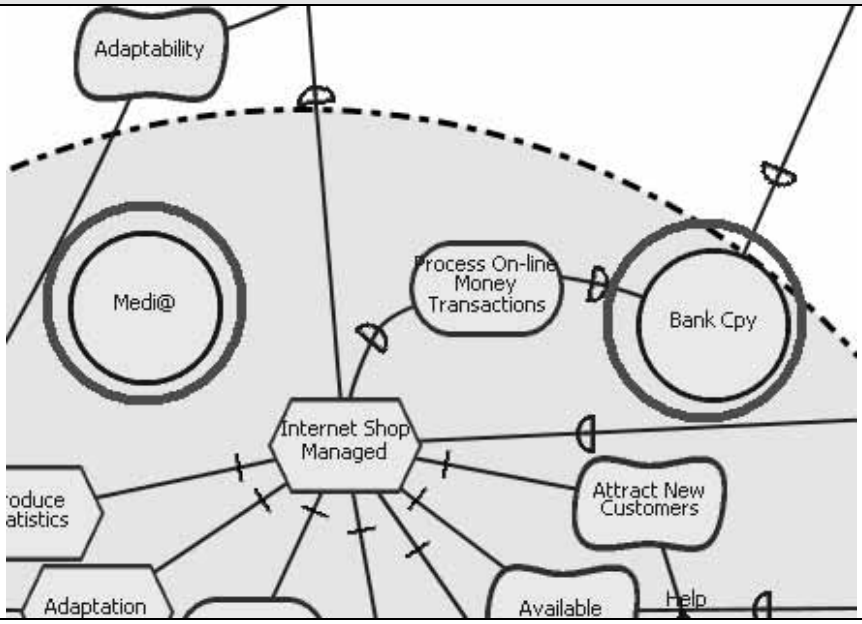


Tabela IV - Softgoals como Goals.

Entrada	04
Descrição	Modelar *Softgoals* como *Goals*
Observação	Os *Softgoals* são *Goals* nos quais não há uma definição clara de satisfação, geralmente representam qualidades ou restrições como segurança, performance, etc. A satisfação dos *Softgoals* pode ocorrer em diferentes níveis, pode ser parcial ou completa. Os *Goals* por outro lado não admitem satisfação parcial.
Referência	[Webster 2005]
Questões	Existe algum *Goal* com adjetivos associados no rótulo? A condição expressa no rótulo do *Goal* pode ser satisfeita parcialmente, ou não pode ser alcançada completamente?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela V - Elementos Intencionais ligados à atores sem dependência.

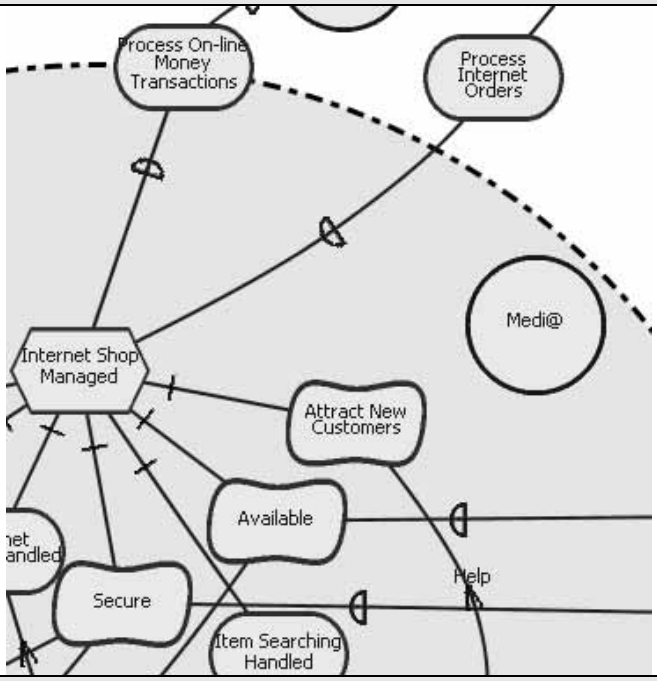
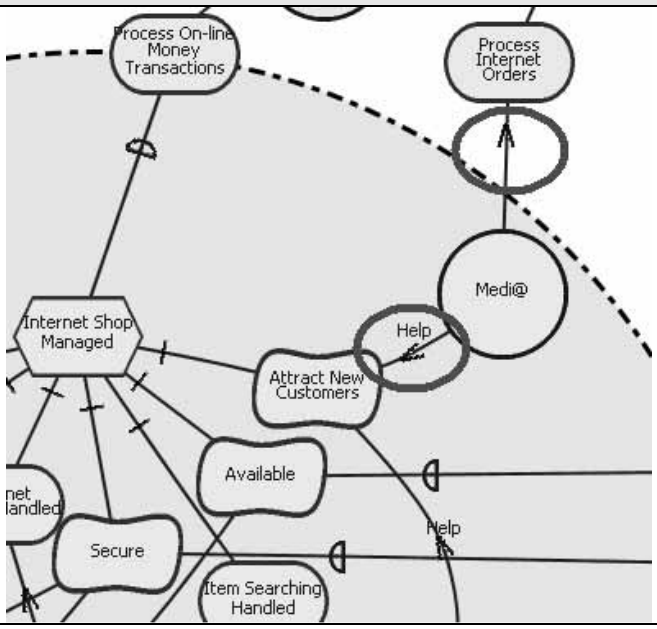
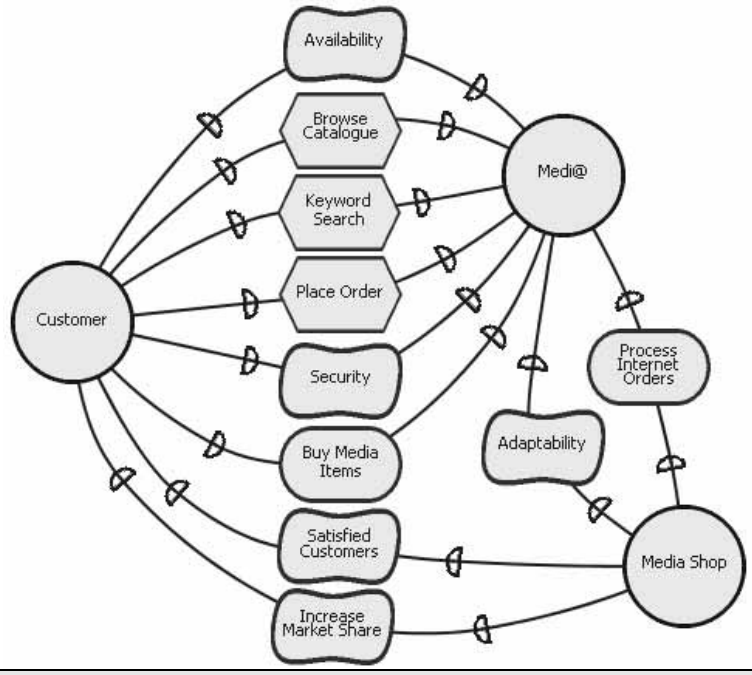
Entrada	05
Descrição	Elementos ligados a atores sem usar dependências
Observação	A única ligação que pode ligar atores à elementos intencionais é a ligação de dependência.
Referência	[Grau 2008]
Questões	Existe algum ator ligado à um elemento intencional sem que seja utilizada a ligação de dependência?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela VI - Dependências sem um dos lados.

Entrada	06
Descrição	Dependências com apenas um dos lados
Observação	Dependências representam um relacionamento ternário, onde cada lado da dependência está um ator (ou elemento interno do ator), e no meio está um dependum.
Referência	[Grau 2008]
Questões	Existe alguma dependência em que um dos lados não está ligado a nenhum ator ou elemento intencional?
Exemplo de uso Correto	



Exemplo de uso Incorreto

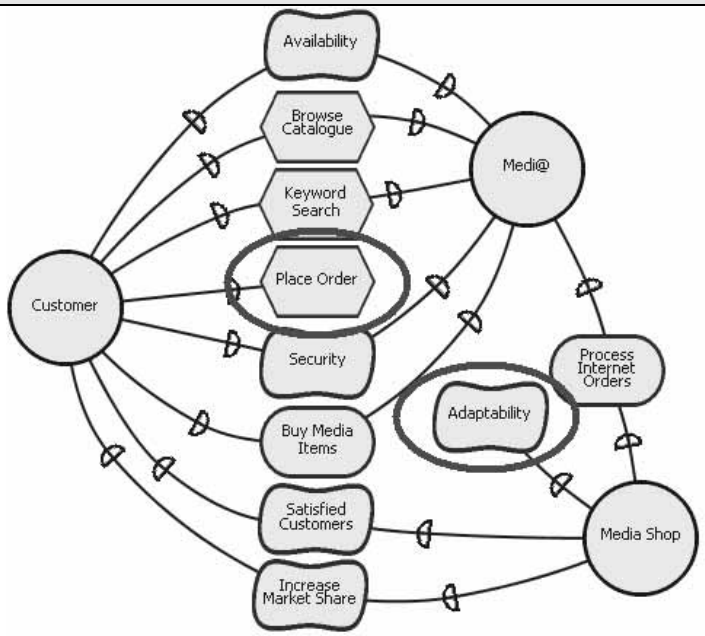


Tabela VII - Dependências que usam outras ligações.

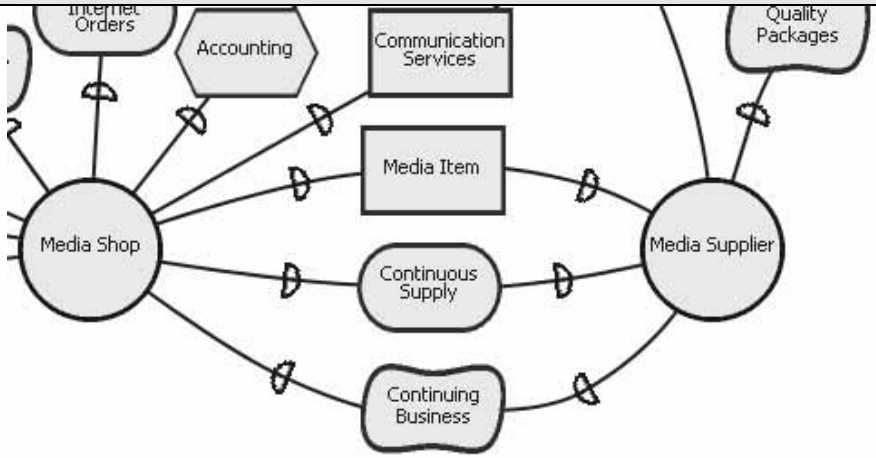
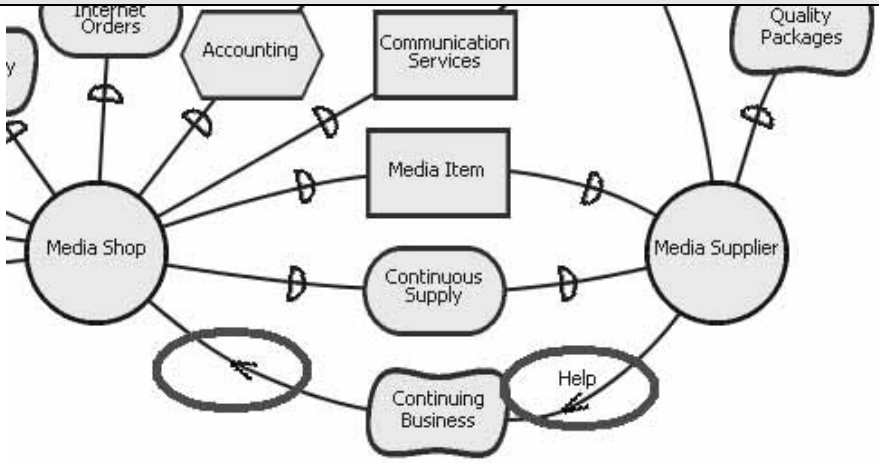
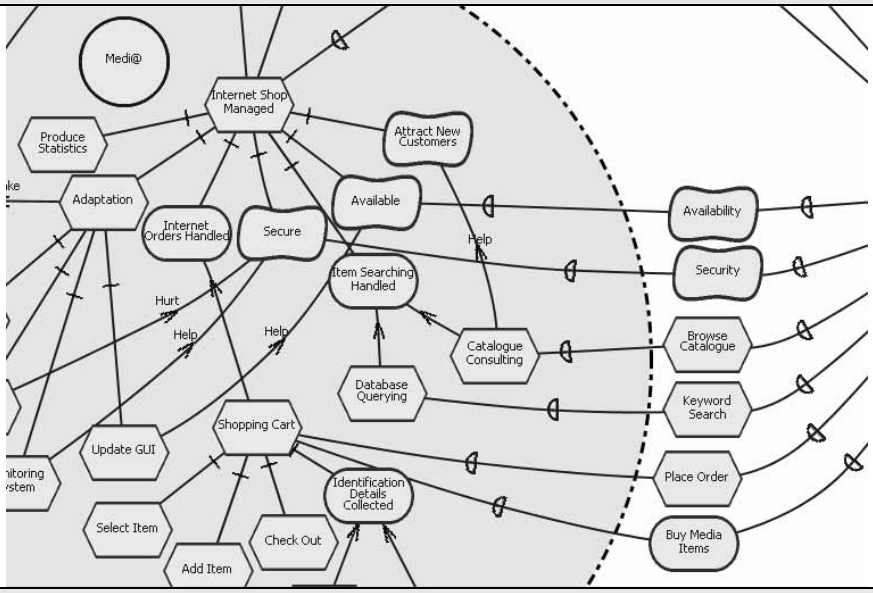
Entrada	07
Descrição	Uso de outros símbolos para indicar dependência
Observação	Relacionamentos de dependência possuem uma ligação própria para representá-lo, outros tipos de ligação estão associados às outras construções.
Referência	[Grau 2008]
Questões	Alguma das dependências apresenta ligações que não utilize o conector de dependência (geralmente representado por um traço com a letra D)?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela VIII - Ligação de Dependência dentro da fronteira do ator.

Entrada	08
Descrição	Usar a ligação de dependência dentro da fronteira do ator
Observação	As ligações de dependência não podem ser utilizadas entre elementos internos de um mesmo ator, ou entre elementos internos e o ator que os contém.
Referência	[Grau 2008]
Questões	Algum elemento interno está ligado à outro elemento interno do mesmo ator por uma ligação de dependência? Algum elemento interno está ligado ao ator que o contém através de uma ligação de dependência?
Exemplo de uso Correto	



Exemplo de uso Incorreto

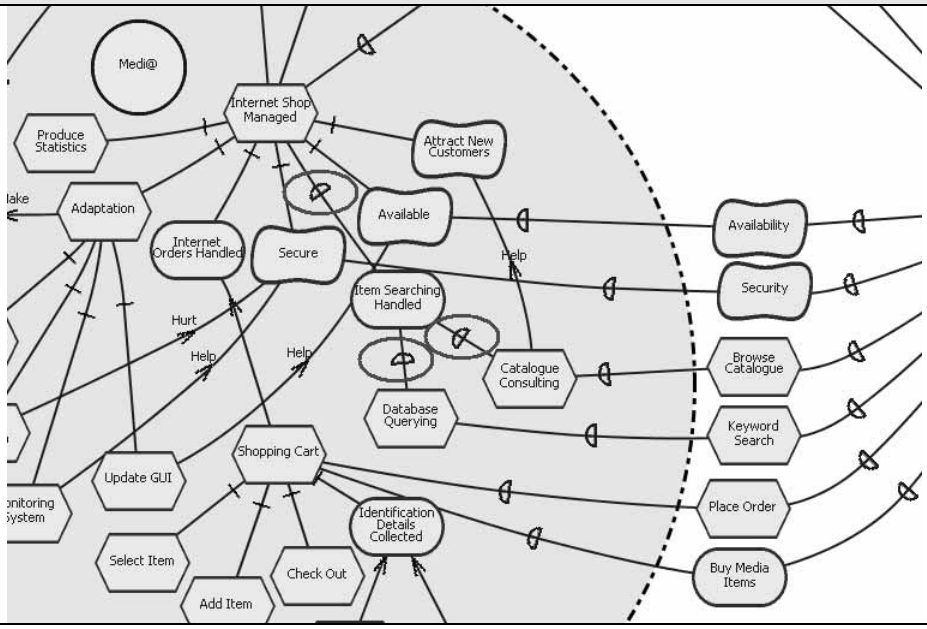


Tabela IX - Usar ligações de dependência sem usar Dependum.

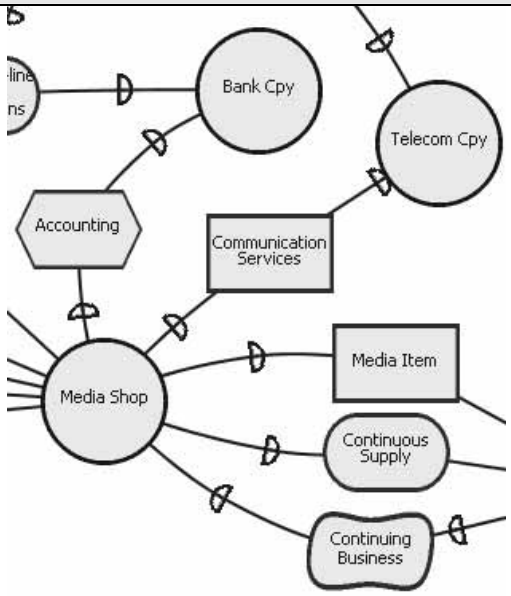
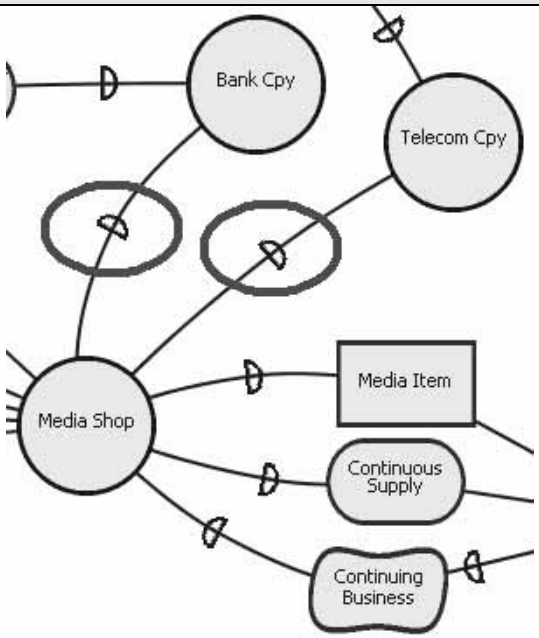
Entrada	09
Descrição	Usar ligação de dependência sem usar um dependum
Observação	O relacionamento de dependência é composto por três partes, dois atores (ou elementos internos nesses atores) e o dependum que é o objeto da dependência.
Referência	[Grau 2005b][Grau 2008]
Questões	Existe alguma ligação de dependência na qual os atores estejam ligados diretamente?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela X - Usar contribuição para Goal.

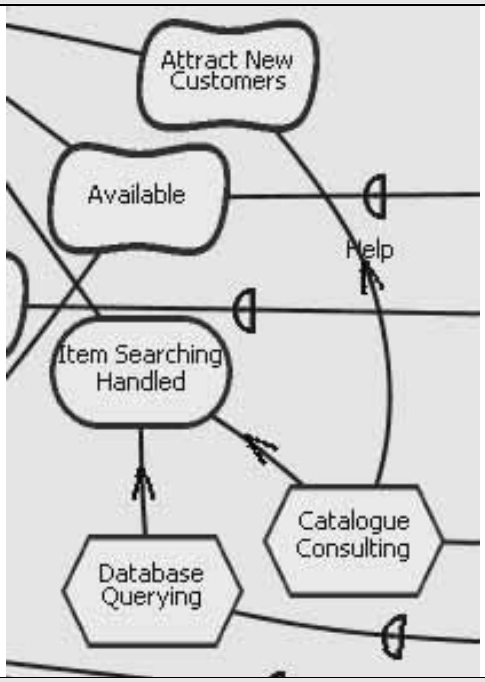
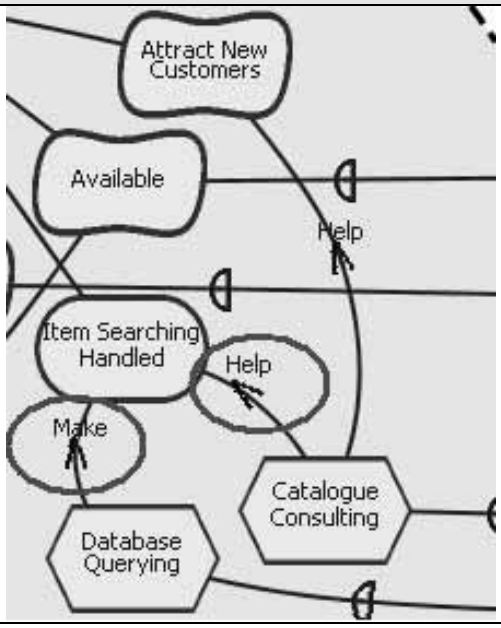
Entrada	10
Descrição	Usar contribuição para Goal
Observação	A ligação de contribuição só aceita como alvo Softgoals
Referência	[Grau 2008]
Questões	Algum Goal recebe ligações de contribuição?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela XI - Goal como Mean.

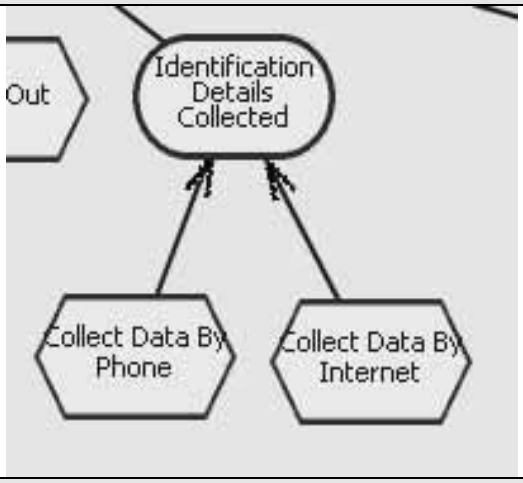
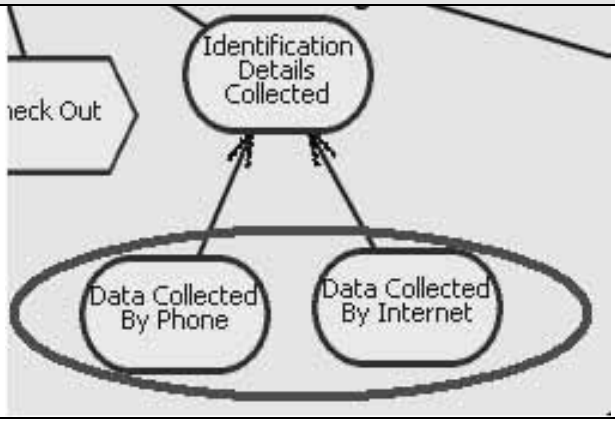
Entrada	11
Descrição	Usar Goal como Mean em uma ligação Means-ends
Observação	O uso de Goals como alternativas para operacionalizar outros Goals não deve ser considerada, para essa finalidade devem ser utilizadas Tarefas.
Referência	[Grau 2008]
Questões	Existe algum Goal como Mean de uma ligação Means-ends? Existe algum ator do lado que sai a seta em uma ligação Means-ends?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela XII - Decomposição entre goals e tarefas.

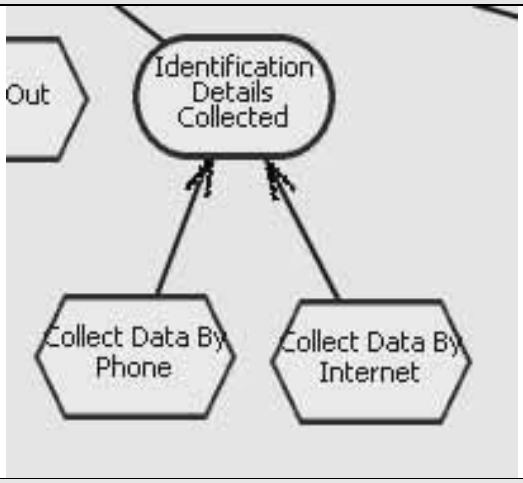
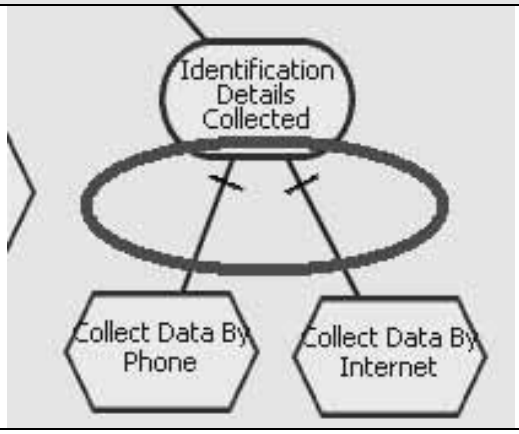
Entrada	12
Descrição	Usar ligações de decomposição entre Goals e tarefas
Observação	Um goal não pode se decompor em outros elementos, no caso das tarefas elas podem se ligar à goals através de ligações de Means-Ends.
Referência	[Grau 2008]
Questões	Existe algum Goal ligado à tarefas através de ligação de decomposição? O Goal está no lado próximo ao traço?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela XIII - Decomposição fora da fronteira do ator.

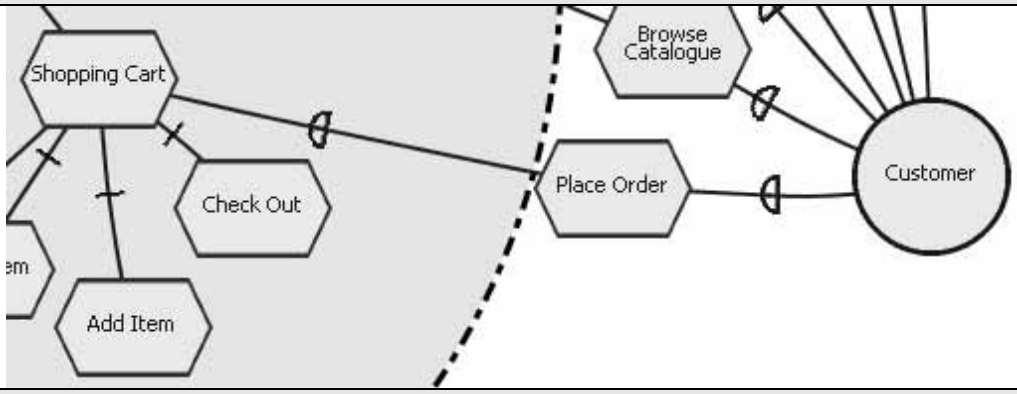
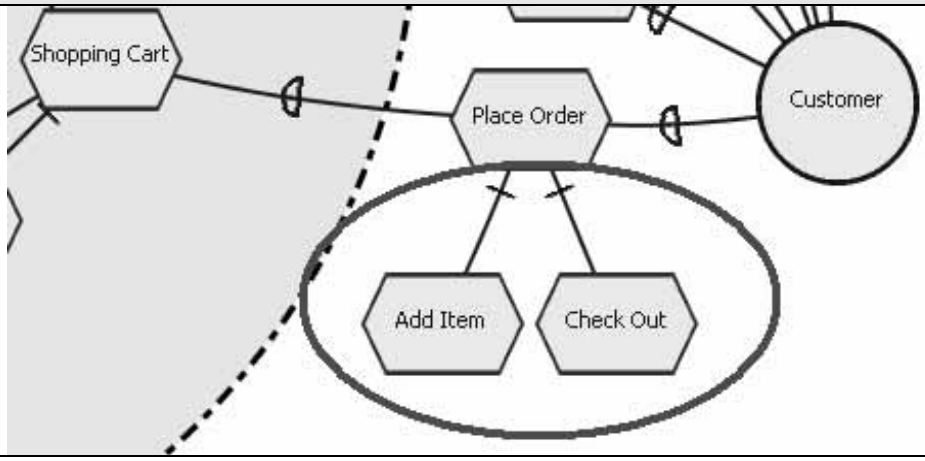
Entrada	13
Descrição	Usar decomposição de tarefas fora da fronteira do ator
Observação	A decomposição de tarefas é feita somente entre elementos internos às fronteiras dos atores.
Referência	[Grau 2008]
Questões	Existe alguma ligação de decomposição fora da fronteira dos atores? Existe alguma ligação de decomposição entre uma tarefa dentro da fronteira do ator e algum elemento fora da fronteira?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela XIV - Means-Ends fora da fronteira do ator.

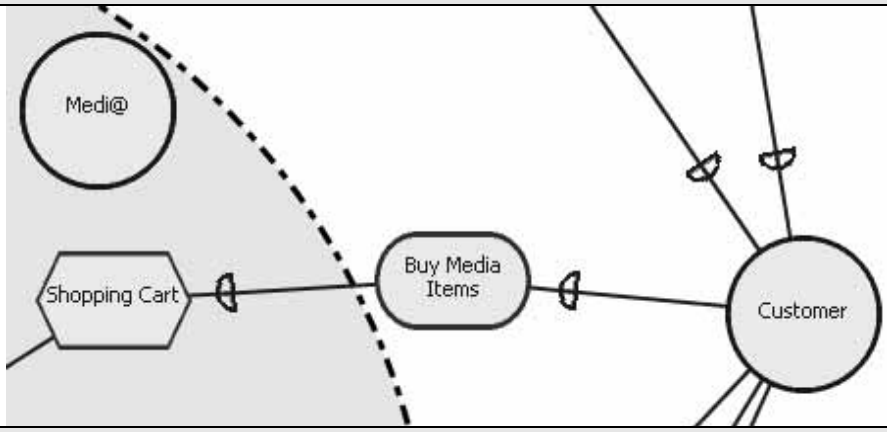
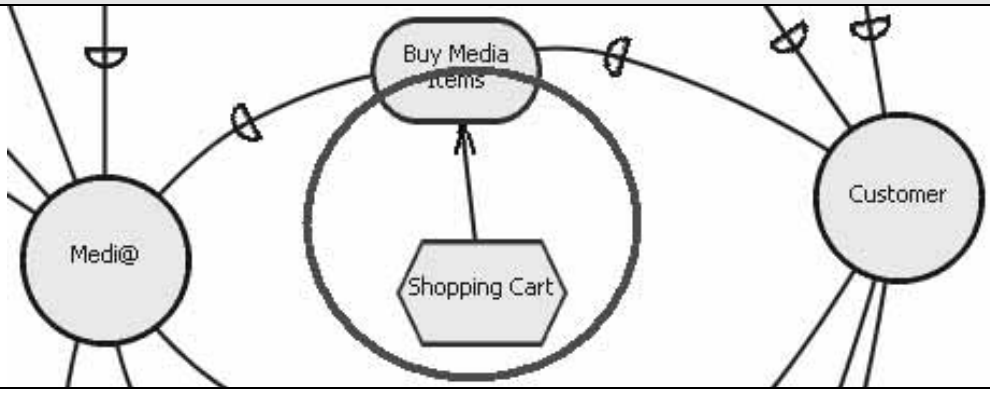
Entrada	14
Descrição	Usar Means-Ends fora da fronteira do ator
Observação	A ligação Means-Ends é feita somente entre elementos internos à fronteira do ator.
Referência	[Grau 2008]
Questões	Existe alguma ligação Means-Ends entre elementos fora da fronteira do ator? Existe alguma ligação Means-Ends entre um elemento interno à fronteira do ator e um elemento externo à ela?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela XV - Contribuição fora da fronteira do ator.

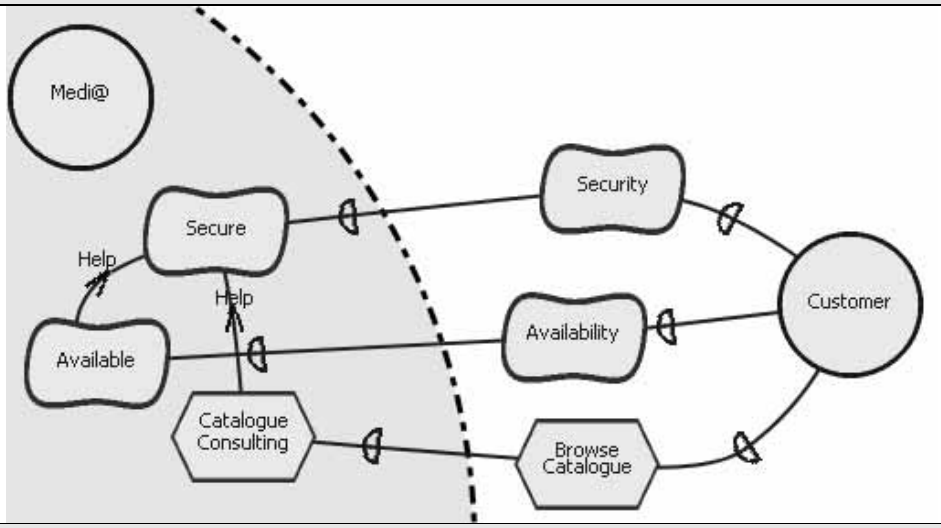
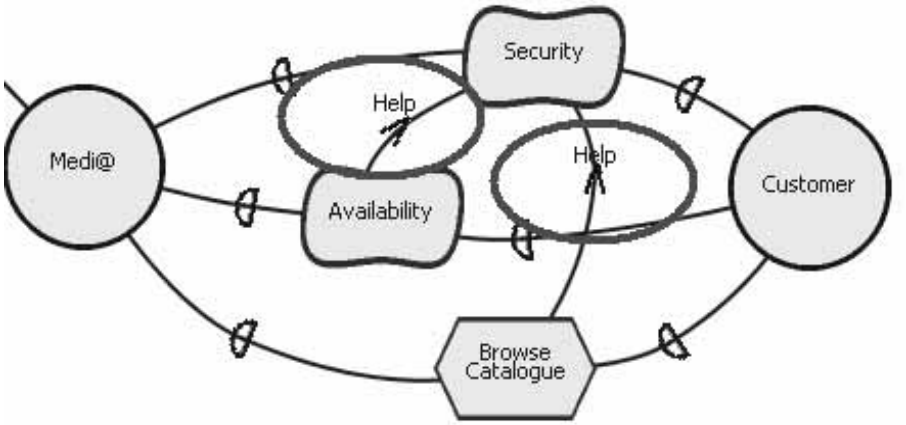
Entrada	15
Descrição	Usar Contribuição fora da fronteira do ator
Observação	A ligação de contribuição é feita somente entre elementos internos à fronteira do ator.
Referência	[Grau 2008]
Questões	Existe alguma ligação de contribuição entre elementos fora da fronteira do ator? Existe alguma ligação de contribuição entre um elemento interno à fronteira do ator e um elemento externo à ela?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela XVI - Circularidade.

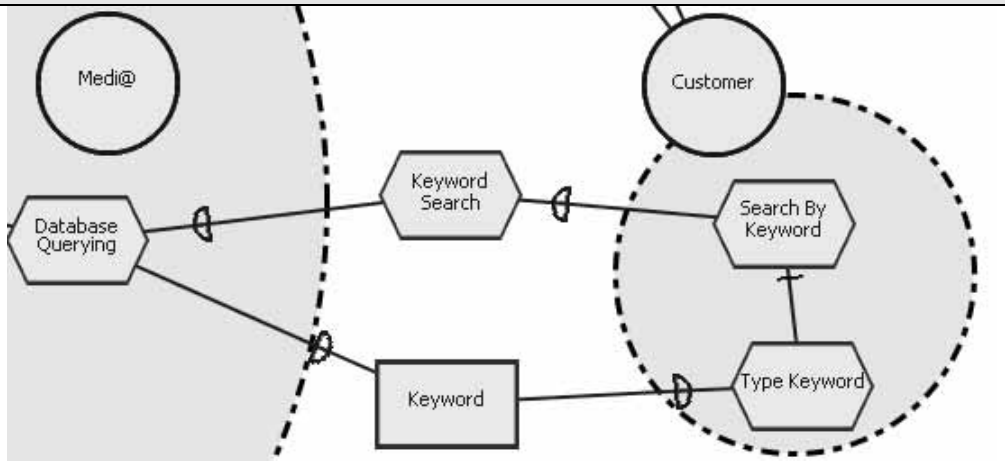
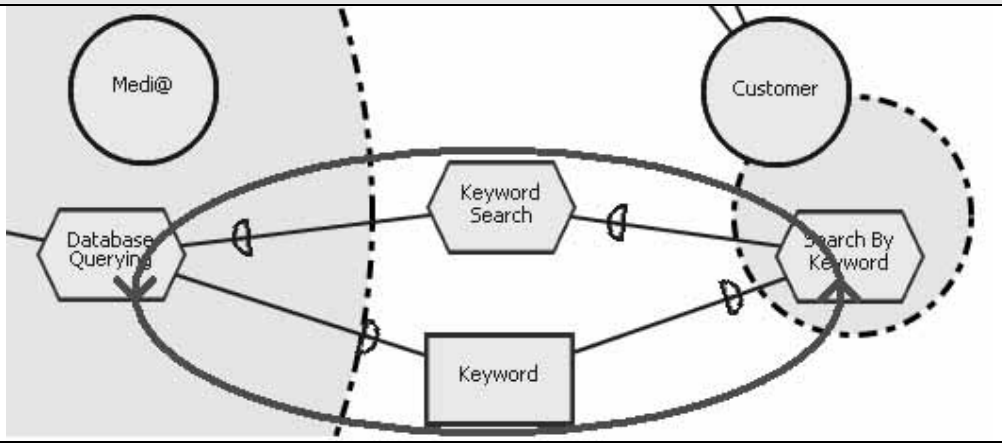
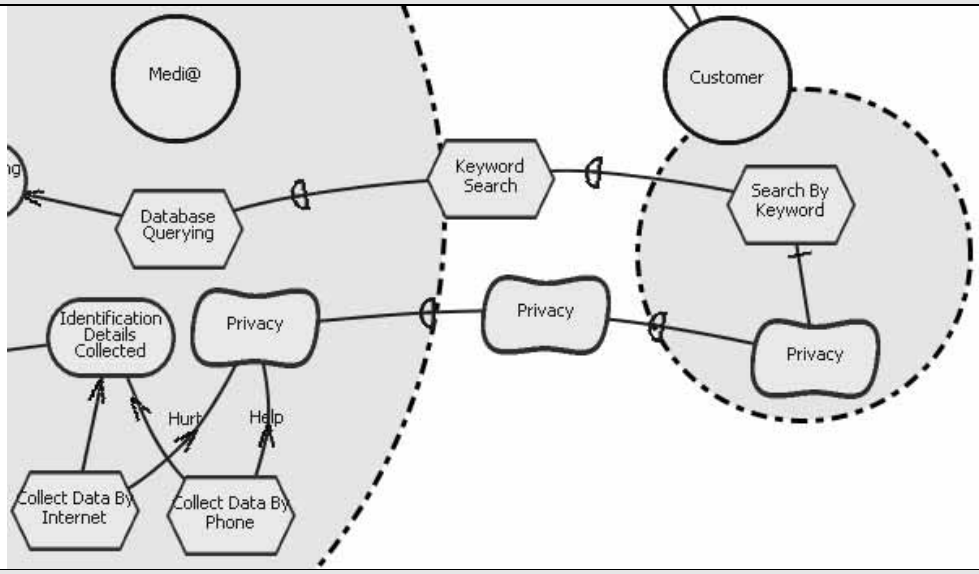
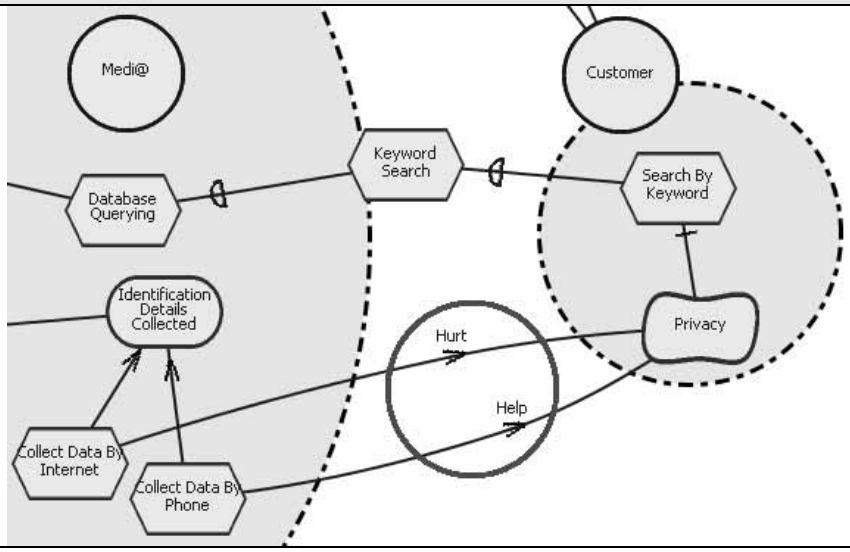
Entrada	16
Descrição	Circularidade
Observação	Os modelos podem conter um ciclo de ligações tendo inícios e fins não óbvios.
Referência	[Horkoff 2006]
Questões	Existem elementos internos de dois atores com dependências nos dois sentidos? Existem Softgoals com contribuições mútuas?
Exemplo de uso Correto	
	
Exemplo de uso Incorreto	
	

Tabela XVII - Usar ligações que passam as fronteiras dos atores.

Entrada	17
Descrição	Usar ligações que passam das fronteiras dos atores
Observação	As únicas ligações que podem se comunicar entrando e saindo das fronteiras dos atores são as ligações de dependência. As outras ligações, como contribuição decomposição e Means-Ends devem à fronteira do ator que está sendo analisado.
Referência	[Horkoff 2006]
Questões	Alguma ligação atravessa as fronteiras dos atores se comunicando diretamente com outros atores ou elementos intencionais dos atores?
Exemplo de uso Correto	



Exemplo de uso Incorreto



Anexo B - i* Guide [Grau 2008]

Este anexo apresenta a versão do guia do i usado como fonte nesta dissertação.*

i* Wiki : iStarGuide

iStarGuide

Version Number	Date	Author(s)
3.0	August, 2007	Samer Abdulhadi
2.0	October, 2006	Jennifer Horkoff, Eric Yu
1.0	August, 2006	Gemma Grau

orthogonal to the distinction between a hard goal and a Softgoal. Discuss
        - 4.2.2.1.3 Guideline (Beginner,Concept) To indicate that a Goal can be achieved by performing several sub-tasks, model the decomposition by introducing a Task. Discuss
        - 4.2.2.1.4 Guideline (Beginner,Concept) Use multiple Means-End Links from Tasks to a Goal to indicate alternatives. Discuss
        - 4.2.2.1.5 Guideline (Beginner,Concept) Don't mix Goals and Tasks in the Means-Ends links. Discuss
        - 4.2.2.1.6 Guideline (Beginner,Naming) Use precise language to name a Goal or a Task. Discuss
        - 4.2.2.1.7 Guideline (Beginner,Concept) A Goal can only be decomposed using Means-Ends Links. Discuss
    - 4.2.2.2 Softgoals
        - 4.2.2.2.1 Guideline (Beginner,Concept) Do not confuse between a Softgoal and a Task. Discuss
        - 4.2.2.2.2 Guideline (Beginner,Notation) Use the proper i* Softgoal notation. Discuss
        - 4.2.2.2.3 Guideline (Intermediate,Concept & Evaluation) Softgoals and Goals should be decomposed. Discuss
    - 4.2.2.3 Tasks
        - 4.2.2.3.1 Guideline (Beginner,Concept) Do not confuse between a Task and a Resource. Discuss
    - 4.2.2.4 Resources
        - 4.2.2.4.1 Guideline (Beginner,Concept) Use a Resource when the Actor asks for the provision of a clearly defined and concrete resource, which can be physical or information entity. Discuss
        - 4.2.2.4.2 Guideline (Beginner,Concept) Model a human or system as a resource only if you want to ignore their goals and intentions. Discuss
    - 4.2.2.5 Beliefs
        - 4.2.2.5.1 Guideline (Beginner,Concept) Describe the effect of Beliefs on Softgoals using Contribution Links. Discuss
    - 4.2.2.6 Guideline (Beginner,Layout) Avoid overlapping elements inside or outside Actors. Discuss
    - 4.2.2.7 Guideline (Beginner,Layout) Connect each Strategic Dependency Link in an SR model to the correct element within the actor. Discuss
    - 4.2.2.8 Guideline (Beginner,Layout) Adopt or follow a consistent direction for the goal refinement/decomposition hierarchy as much as possible. Discuss
    - 4.2.2.9 Guideline (Beginner,Layout) Do not draw SR model elements outside the boundaries of the corresponding actors. Discuss

1 Purpose

1.1 Education

The i* Guide is intended to be both an introduction to i* for new users and a reference guide for experienced users. It is constituted of basic i* notation and a glossary and provides relatively elaborated material on the usage of the i* framework and notation through introducing guidelines that are associated with discussions and illustrations. These Guidelines, which touch on some of the fundamental concepts of i* Framework, are intended to, among other desirable modeling characteristics, improve the readability, understandability, usability, and feasibility of i* models and enhance the overall consistency and effectiveness of the i* modeling process. Therefore, the net positive effect is to reduce variation in practice among users of i* modeling framework. Furthermore, the i* Guide refers the readers to original sources where more advanced, detailed, or thorough material and discussions can be found.

1.2 Collaboration

As this guide is intended to be part of the i* collaborative wiki we encourage feedback and collaboration in terms of suggesting alternative i* syntax and semantics, including extensions to the Framework. However, in order to keep the Guide brief and easily understandable, we suggest collecting i* adaptations, expansions, and further examples in separate document(s), namely the istarSyntaxVariations page. If enough variations of i* usage are collected, this can facilitate interesting and useful comparisons.

Moreover, and to facilitate this collaboration process, i* wiki will host two versions of the i* Guide: A stable version and an open version. The open version is accessible to all registered i* wiki users to comment and provide suggestions on individual guidelines. Therefore, it is highly appreciated and valued that i* community utilizes this open version as a basis to develop and enhance the i* Guide further. In effect, you can think of the open version as a prototype that invokes and generates further input toward the achievement of a better artifact. This link connects you to the individual wiki pages of the guidelines. Once connected, you will see a wiki page containing a list of Level and Type guidelines' categories (defined in the next Section 2, "Using the Guidelines") that lead you to the individual guidelines' pages. For example, if you click on Beginner under Level, you will see a list of all the guidelines categorized under Beginner Level. To access the page of any guideline in the list, just click on the guideline and the guideline page will open. You can use this guideline page for comments and discussions.

2 Using the Guidelines

Each guideline is annotated with a set of attributes of the form (Level, Type), where Level is: Beginner, Intermediate, or Advanced, and Type is: Concept, Naming, Notation, Layout, Methodology, or Evaluation.

If a guideline is given more than one Category attribute, this means this guideline probably spans more than one Category. The objective of explicitly annotating the guidelines with these attributes is to help the users of the i* Guide to browse through or retrieve the guidelines based on any dimension of these attributes. This annotation also gives i* concerned researchers an idea about the current distribution and coverage of the guidelines over the spectrum of the annotated attributes.

The following is an explanation of each of the terms pertaining to Level and Category attributes:

Guidelines' Level attribute:

Level	Description
Beginner	This level assumes that the modeler is a new user of i* modeling framework and has not been exposed to it previously
Intermediate	This level assumes that the modeler has been exposed to and used i* modeling framework to some extent and passed the level of violating fundamental i* guidelines
Advanced	This level is for modelers who acquired substantial knowledge about using and applying i*

Guidelines' Type attribute:

Type	Description
Concept	deals with the arrangement and organization of i* models and the way the contents of the models appear and are placed. It also covers issues related to modeling space and complexity.
Naming	deals with how the actors, links, and elements are named. It also covers issues related to using colors and other non-conventional icons.
Notation	deals with the proper selection and use of i* notation such as elements, actors, and links.
Layout	deals with the arrangement and organization of i* models and the way the contents of the models appear and are placed. It also covers issues related to modeling space and complexity.
Methodology	deals with the procedural choices or approaches that can help in attaining an overall systematic modeling process.

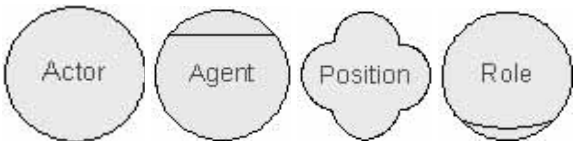
Evaluation	deals with using i* models to evaluate the satisfaction or denial of Actors' intentional elements and strategic dependencies. Violating Evaluation guidelines could lead to incorrect or weak reasoning about the rational behind the satisfaction or denial of Actors' intentional elements and strategic dependencies.

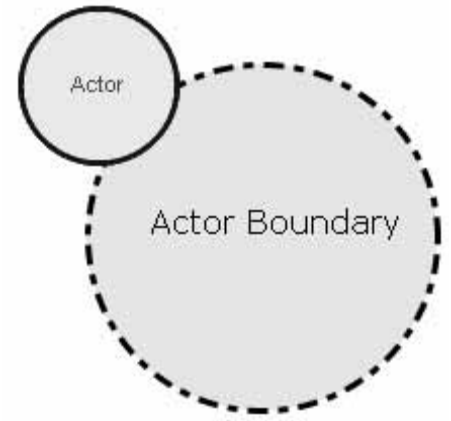
Also, the Guide presents both i* Glossary and the related Guidelines in section 4 to better organize the Guide and help the reader relate between the presented glossary and the associated guidelines more effectively and efficiently. The glossary provides the definitions for i* notation, vocabulary, and modeling language. The guidelines proceed after the glossary to provide recommendations, discussions, illustrations on the use of the i* modeling language. Therefore, in order to maximize the understandability and the return from using the Guide, it is recommended that the new i* users get acquainted first with the glossary and fundamental definitions of i* before proceeding to the guidelines and their associated discussions.

3 Summary of i* Notation

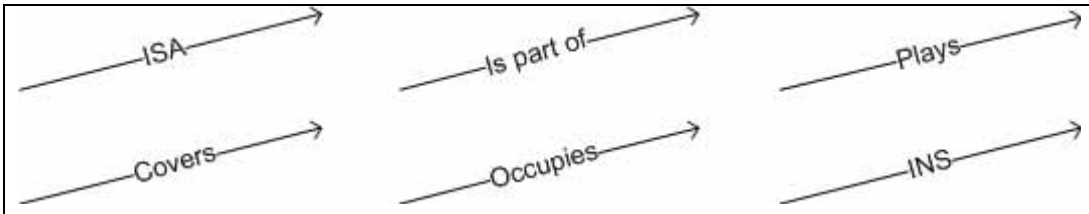
This section provides only the graphical notation of i* syntax. An explanation of each notation can be found in the i* Glossary and Guidelines Section. Note that as i* models can be created by a variety of software tools, there can be small variations in notation appearance, mainly pertaining to color and line size.

Actors

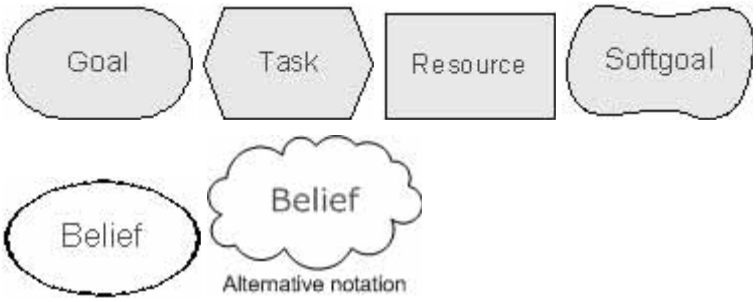




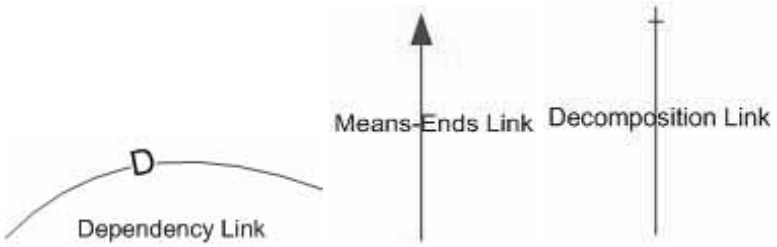
Actor Associations



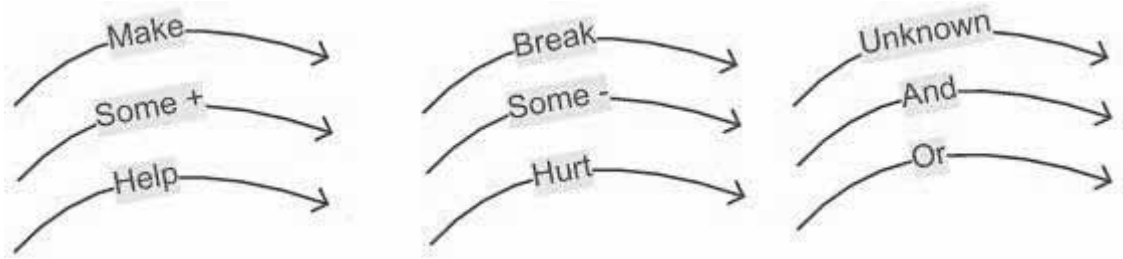
Elements



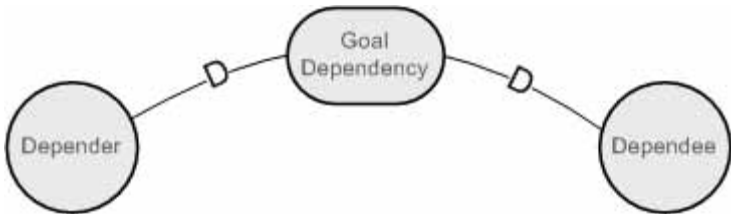
Links

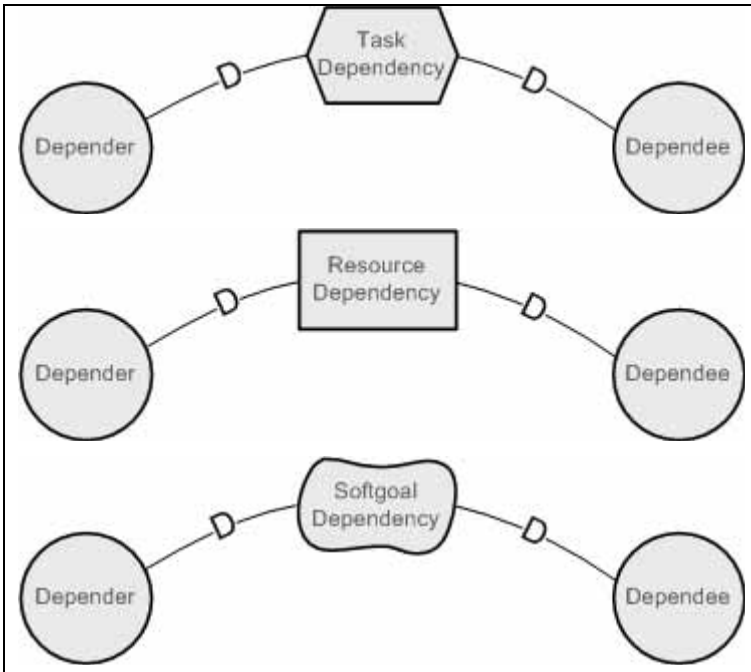


Contribution Links

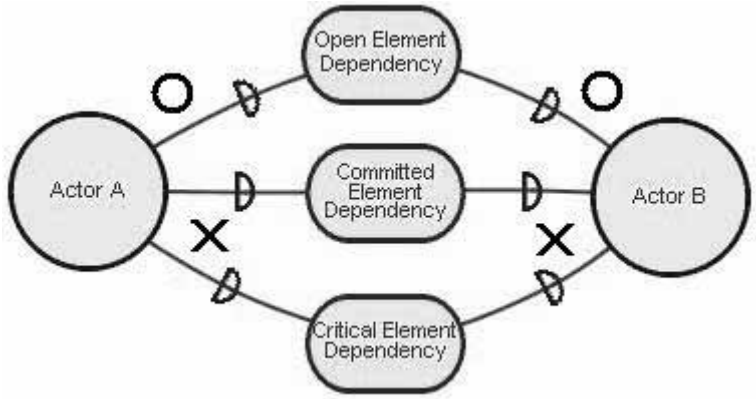


Strategic Dependencies

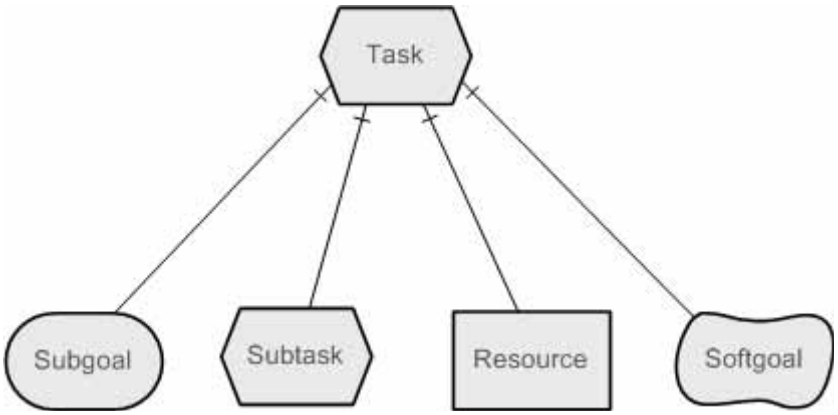




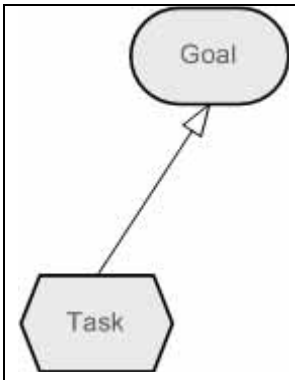
Dependency Strengths



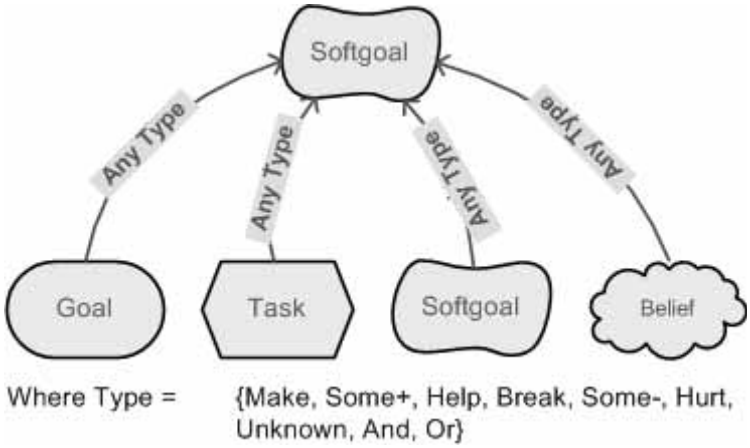
SR Decomposition Links



SR Means-End Links



Contributions



4 i* Glossary and Guidelines

4.1 Strategic Dependency (SD) Model

Set of nodes and links where each node represents an actor and each link between two actors indicates that one actor depends on the other for something in order that the former may attain some goal. The SD model is used to express the network of intentional, strategic relationships among actors. SD diagrams depict the strategic dependencies between Actors, but do not depict the internal rational behind these dependencies.

4.1.1 Actors

Active entities that carries out actions to achieve goals by exercising its know-how. We use the term actor to refer generically to any unit to which intentional dependencies can be ascribed. Agents, roles and positions are sub-units of a complex social actor, each of which is an actor in a more specialized sense.

4.1.1.1 Role

Abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor. Its characteristics are easily transferable to other social actors. The dependencies associated with a role apply regardless of the agent who plays the role.

4.1.1.2 Agent

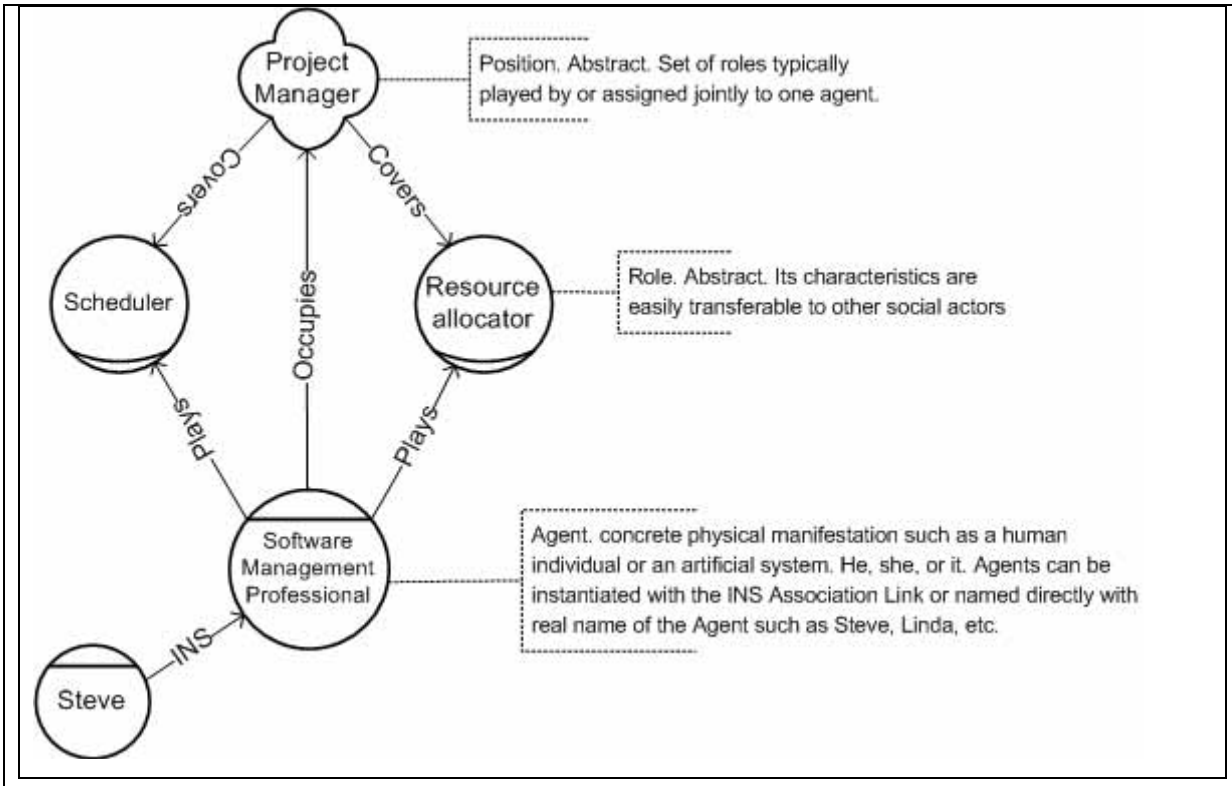
Actor with concrete, physical manifestations, such as a human individual. We use the term agent instead of person for generality, so that it can be used to refer to human as well as artificial (hardware/software agents). An agent has dependencies that apply regardless of what roles he/she/it happens to be playing. These characteristics are typically not easily transferable to other individuals, e.g. its skills and experiences, and its physical limitations.

4.1.1.3 Position

Intermediate abstraction that can be used between a role and an agent. It is a set of roles typically played by one agent (e.g., assigned jointly to that one agent). We say that an agent occupies a position. A position is said to cover a role. A position is abstract and an amalgamation of roles. A position can be ascribed to a human or non-human, even though the later case is rare.

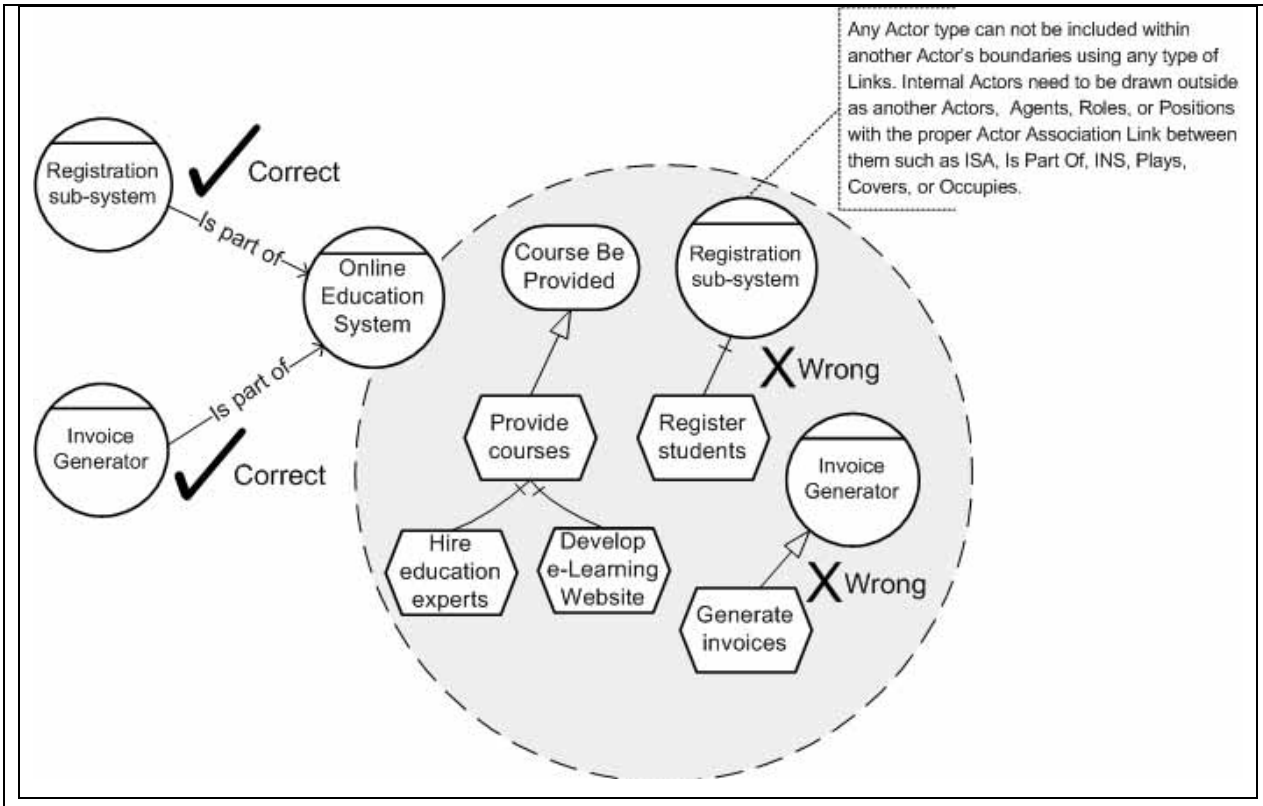
4.1.1.4 Guideline (Beginner,Concept) Use Agents/Roles instead of actors when the distinction is easily made. Discuss

Discussion: Actors are the general type, whereas, agents and roles are more specific. Therefore, it would be more accurate and not superfluous to model Actors as Agents or Roles when the distinction is clear between them. Some trade offs between using the general Actor notation as opposed to the specialized Actor notation are discussed in Section 4.5.1 “Agents, Roles, and Positions”.



4.1.1.5 Guideline (Beginner,Concept) Do not include an Actor within another Actor. Discuss

Discussion: Actors are active and autonomous entities that should be modeled separately. "Sub-system" in the illustration can be modeled as actors that have Dependency Links with the main system and/or other actors. They can also be modeled with Association Links such as “is-part-of” and “ISA”, to the higher-level system.

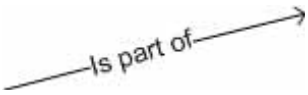


4.1.2 Actor Association Links

The relationships between actors are described by graphical association links between actors.

4.1.2.1 Is-part-of (Part) Association

Roles, positions, and agents can each have subparts. Aggregate actors are not compositional with respect to intentionality. Each actor, regardless of whether it has parts, or is part of a larger whole, is taken to be intentional. There can be intentional dependencies between the whole and its parts, e.g., a dependency by the hole on its parts to maintain unity.



4.1.2.2 ISA Association

The is_a association represents a generalization, with an actor being a specialized case of another actor. Both ISA and Is-part-of can be applied between any two instances of the same type of actor.

4.1.2.3 Plays Association

The plays association is used between an agent and a role, with an agent playing a role. The identity of the agent who plays a role should have no effect on the responsibilities of that role, and similarly, aspects

of an agent should be unaffected by the roles it plays.

4.1.2.4 Covers Relationship

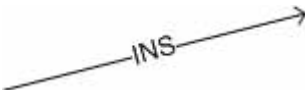
The association link covers is used to describe the relationship between a position and the roles that it covers.

4.1.2.5 Occupies Relationship

The occupies link is used to show that an agent occupies a role, meaning that it plays all of the roles that are covered by the position.

4.1.2.6 INS Relationship

The ins association, representing instantiation, is used to represent a specific instance of a more general entity. An agent is an instantiation of another agent.



4.1.2.7 Guideline (Beginner,Concept) Use 'ISA' and "Is part of' Association Links only between actors of the same type. Discuss

Discussion: Role, Agent, and Position are autonomous social actors of different types, which characterize different levels of abstraction. This means that a Role, for example, can be part of another Role, but cannot be part of an Agent or Position. This is true also for Agent and Position. As explained in Guideline 4.1.1.4, a Role, however, can be associated with an Agent or Position using the Plays and Covers Association Links respectively. An Agent can be associated with a Position using the Occupies Association Link. Refer to Section 4.1.1 “Actors” and Section 4.1.2 “Actor Association Links” for definitions of various types of actors and association links.

Figure 1 depicts correct scenarios, which are true also for Agent, and Position. Figure 2 depicts incurrent or wrong scenarios that involve Roles, Agents, and Positions. Figure 3 depict scenarios that are allowed in incomplete (in progress) models as the available information to the modeler about the type (Role, Agent, or Position) of the general ‘Actor’ is not yet known. Once the information becomes available, the correct actor type should be used.

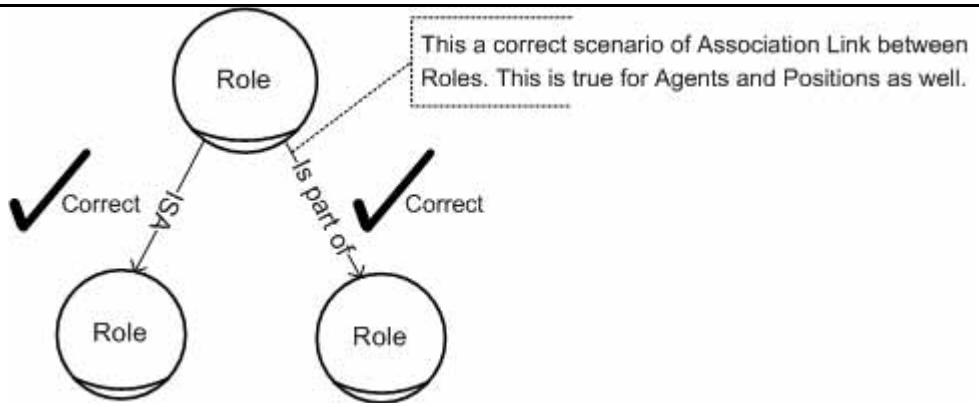


Figure 1 Correct scenarios of using Association Links

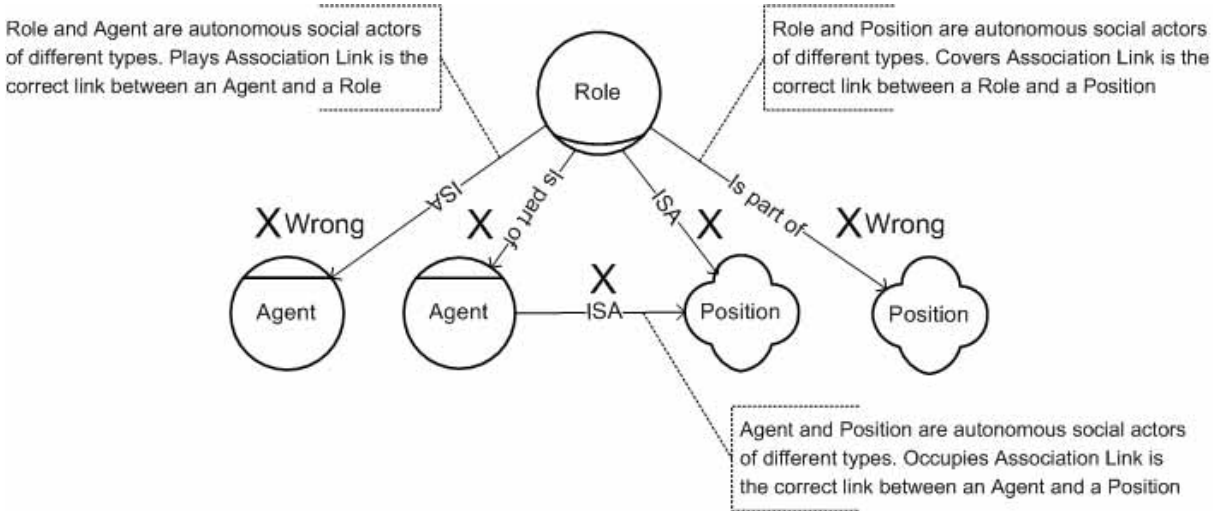


Figure 2 Incorrect scenarios that involve Roles, Agents, and Positions

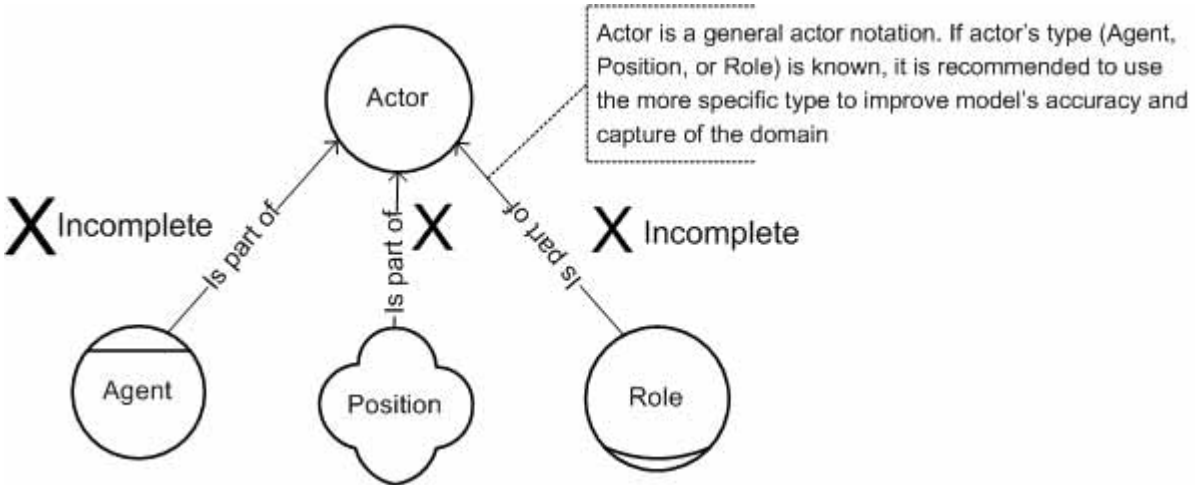


Figure 3 Scenarios that are allowed in incomplete (in progress) models

4.1.3 Strategic Dependencies

Dependee

Actor who is depended upon on a dependency relationship.

Depender

The depending actor on a dependency relationship.

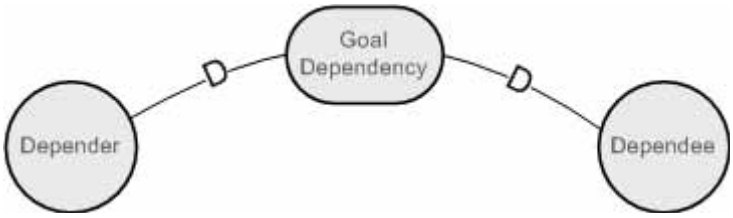
Dependum

Element around which a dependency relationship centers.

We distinguish among four types of dependencies, based on the type of the dependum: Resource dependency, Task dependency, Goal dependency, Softgoal dependency.

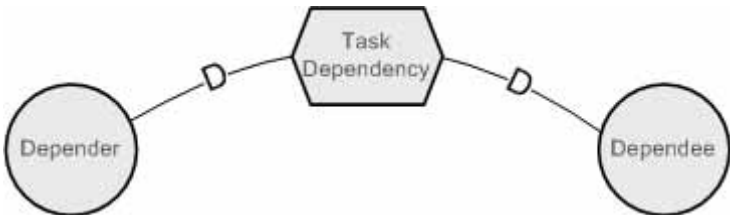
4.1.3.1 Goal Dependency

In a goal dependency, the depender depends on the dependee to bring about a certain state of affairs in the world. The dependum is expressed as an assertion statement. The dependee is free to, and is expected to, make whatever decisions are necessary to achieve the goal (namely, the dependum). The depender does not care how the dependee goes about achieving the goal.



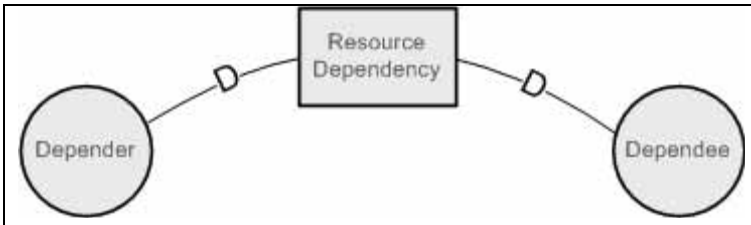
4.1.3.2 Task Dependency

In a task dependency, the depender depends on the dependee to carry out an activity. The dependum names a task which specifies how the task is to be performed, but not why. The depender has already made decisions about how the task is to be performed. Note that a task description in i* is not meant to be a complete specification of the steps required to execute the task. It is a constraint imposed by the depender on the dependee. The dependee still has freedom of action within these constraints.



4.1.3.3 Resource Dependency

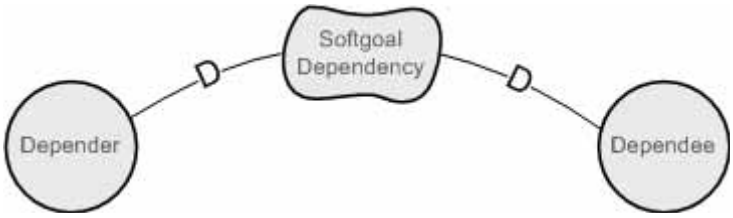
In a resource dependency, the depender depends on the dependee for the availability of an entity (physical or informational). By establishing this dependency, the depender gains the ability to use this entity as a resource. A resource is the finished product of some deliberation-action process. In a resource dependency, it is assumed that there are no open issues to be addressed or decisions to be made about the provision or achievement of the Resource entity.



4.1.3.4 Softgoal Dependency

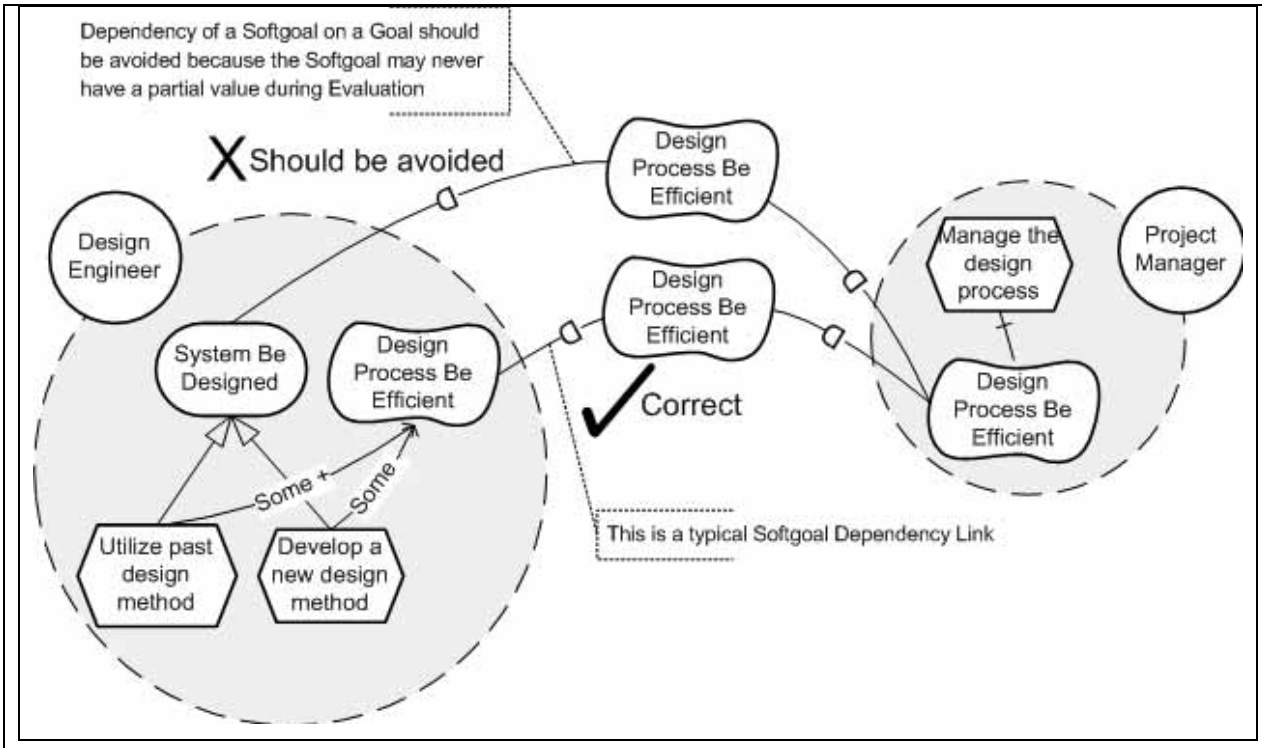
In a softgoal dependency, a depender depends on the dependee to perform some task that meets a softgoal. A softgoal is similar to a goal except that the criteria of success are not sharply defined a priori. The meaning of the softgoal is elaborated in terms of the methods that are chosen in the course of pursuing the goal. The depender decides what constitutes satisfactory attainment of the goal, but does so with the benefit of the dependee's know how. Based on all the concepts presented in Section 1.3.1.3 "Strategic Dependencies", the modeler has the choice of using a Task, Resource, Goal, or Softgoal Dependency Link between actors in a model depending on the context of the design. Each case has different purpose and interpretation. For example, using a Task Dependency Link between two actors means that one of these actors actually depends on the other actor to satisfy and perform the task in a particular way or with some freedom given a set of constraints. Therefore, the task is delegated to another actor with minimum or no freedom of choice. On the other hand, using a Goal or Softgoal Dependency Link means that the Depender actually gives more freedom in choosing which methods to employ to satisfy the dependency or accomplish the Goal or Softgoal.

Additionally, some literature refers to Softgoals as Non-Functional Requirements (NFRs). There are examples, however, of non-functional requirements that are not softgoals. Anything that is quantified is not a softgoal, but can be an NFR. For example, the system must process an order within 2 seconds, an NFR, but not a softgoal. Despite this debate whether Softgoals and NFRs are the same thing, it is still acceptable for beginner users of i* Guide to assume that the term NFR constitutes the same concepts discussed above in regards to Softgoals.



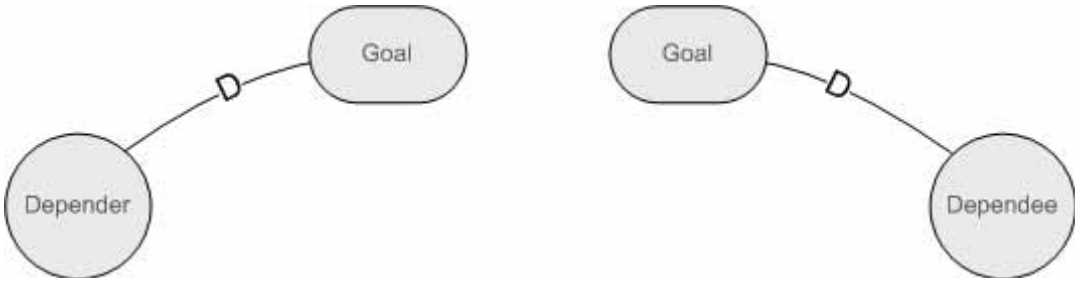
4.1.3.4.1 Guideline (Beginner,Concept) **Softgoal Dependency should not be met directly by a Goal.**
Discuss

Discussion: Do not make a Softgoal depend on a goal because the Softgoal may never have any partial value during evaluation. Please refer to Section 4.6.5 for a complete introductory discussion about Evaluation and Propagation. A typical Softgoal Dependency Link is shown in the illustration.



4.1.3.5 One-Side Dependency

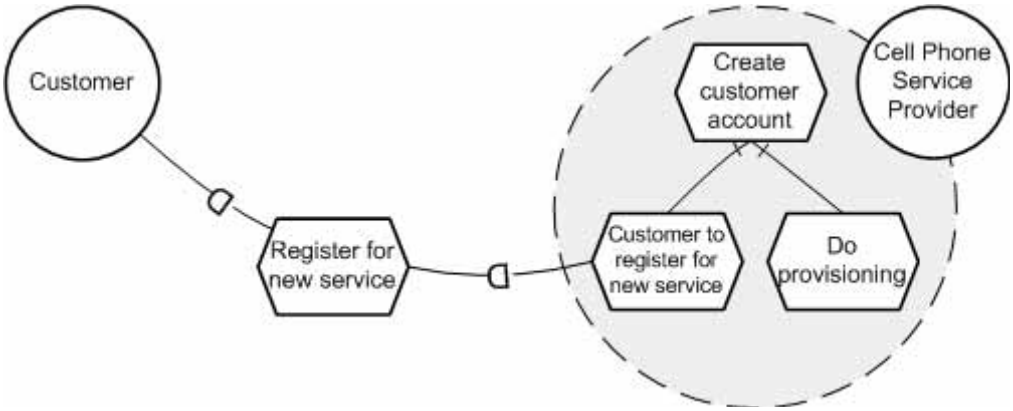
As mentioned in Section 1.4.1.3, Strategic Dependency involves a Depender, Dependum, and Dependee. There are cases, however, where a strategic dependency is modeled using One-Side Dependency, which could be established from the Depender to the Dependum or from the Dependum to the Dependee. In the former case, the Depender wants Goal to be achieved, but no Dependee is available. In the latter case, the Dependee has the ability to achieve Goal, but there is no Depender for that Goal



4.1.3.6 Guideline (Intermediate,Concept & Evaluation) Use the Dependency link to indicate a Strategic Dependency relationship between two Actors Discuss

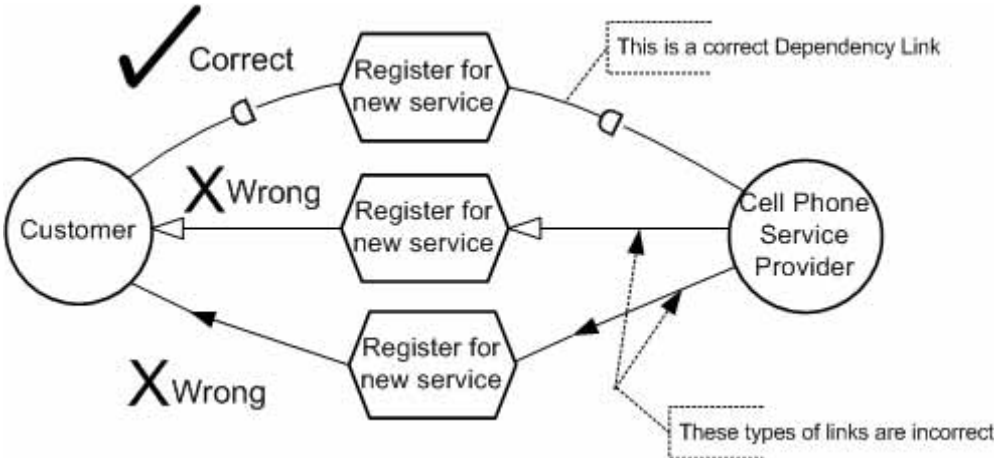
Discussion: The Strategic Dependency Link is used to denote some type of dependency such as, Goal, Softgoal, Task, or Resource dependency, between a Depender and a Dependee. Often, a Dependum is the result of a Means-Ends or Task Decomposition relationship inside the Depender. In these cases, the Means-Ends or Task Decomposition Link should be used inside the Depender, in the SR model.

Strictly speaking, some version of the Dependum (Register for new service) should also appear inside the Depender as shown in the illustration. This is significant for goal evaluation procedure. Please see Section 4.6.5 for a complete introductory discussion about using i* models for evaluation.



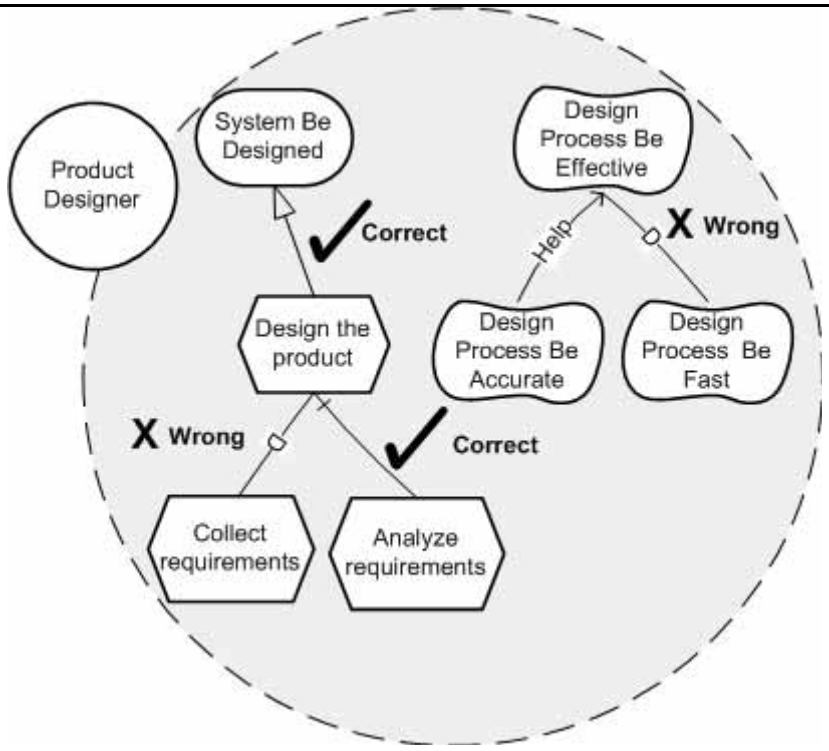
4.1.3.7 Guideline (Beginner,Notation) **Use the "D" symbol notation to denote a Dependency Link. Do not use other symbols such as arrow heads, which can be confused with other types of links. Discuss**

Discussion: Using other notations such as the Means-Ends Link, Contribution Link, or any other arrow head symbols creates inconsistencies, which could lead to model comprehensibility issues.



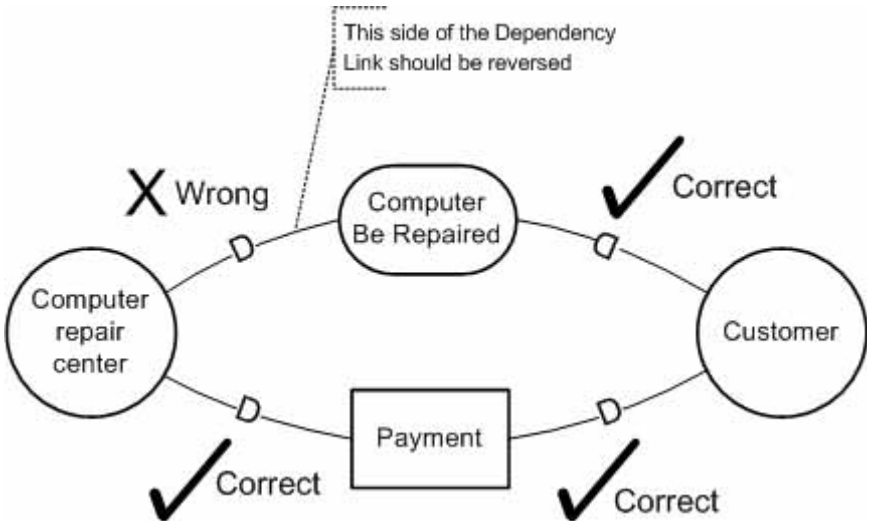
4.1.3.8 Guideline (Beginner,Concept) **Do not use Dependency Links inside an Actor. Discuss**

Discussion: Task element should be broken down, decomposed, or refined into lower level elements using the Decomposition Link, not the Dependency Link. Softgoal element can be broken down, decomposed, or refined into lower level elements using any type of Contribution Link. Contribution links are: Make, Some+, Help, Break, Some-, Hurt, Unknown, And, or Or depending on the context of the analysis.



4.1.3.9 Guideline (Beginner,Concept) **Ensure that both sides of a Dependency Link point in the same direction. Discuss**

Discussion: The direction of the Dependency Link should be consistent in both sides of the Dependum. The Depender depends on the Dependee to satisfy a Dependum. The "D" notation needs to be drawn at both sides of the Dependency Link to show the direction of the dependency.



4.1.3.10 Guideline (Intermediate,Concept & Evaluation) **Do not reuse Dependums in more than one Dependency Relation. Discuss**

Discussion: Dependency Link should have only one link segment on both sides of the Dependum: one on the Depender side and the other one on the Dependee side. Figure 1 depicts a split scenario where the Dependency Link splits into two links from one Dependum to two Dependees. This illustration says that “Patient” depends on both the “Government” and “Other health service providers” to fulfill or satisfy the Goal of, “Health Service Be Provided”. This notation violates the conventional Dependency Link in i* because it makes it difficult for the modeler and analyst to assess and evaluate the satisfaction of the Dependum when it is linked to two different Dependees. Because of the autonomicity of the Actors (Dependees) in i*, every Actor is evaluated separately in terms of its (her or his) contribution to the fulfillment and satisfaction of the Dependum. If the intention of the modeler is AND, the model should have Task Decomposition inside the Depender. The Dependums for the two Dependees will most likely by different. Using the split and join scenarios, however, are acceptable for drat models. Figure 2 depicts a join scenario where two Dependency Links from two Dependers join on one link from one Dependum to one Dependee.

The illustration shows both Dependers, “Patient” and “Citizen”, depend on the Dependee, “Hospital” to satisfice the Softgoal, “Availability [Health Service]”. This non-conventional i* notation, which has been used in one of the Tropos Methodology examples, follows the same lines of reasoning for the split scenario. The degree of satisfaction and Softgoal fulfillment for each Depender might have a different view whether the contribution of the Dependee can really satisfice the Softgoal of both Dependers.

Figure 3 illustrates another ambiguous scenario where two Dependency Links join on and split from the same Dependum. The two Dependums, “Medicine” and “Medicine’s Details”, are two different things that should not be represented by the one Dependum, “Medicine”. Using the proper Dependency Links enhances the accuracy of the model during model analysis and evaluation.

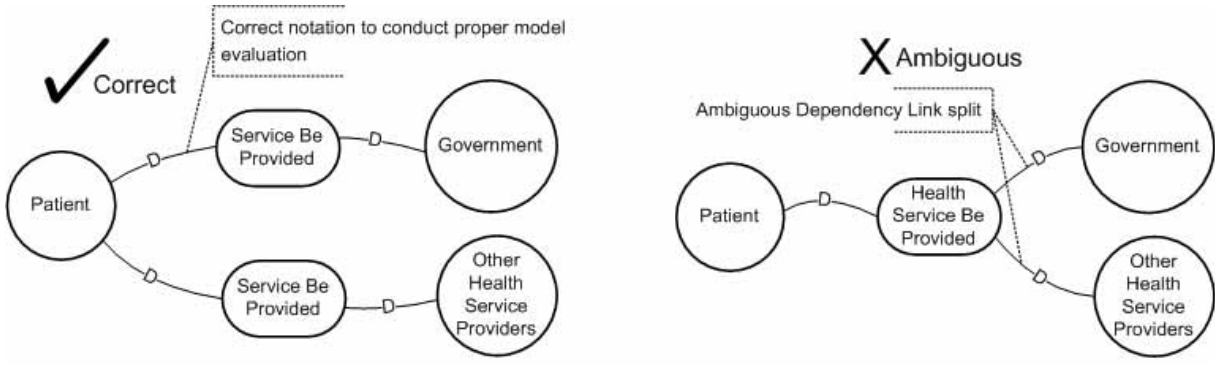


Figure 1 Split scenario of a Dependency Link

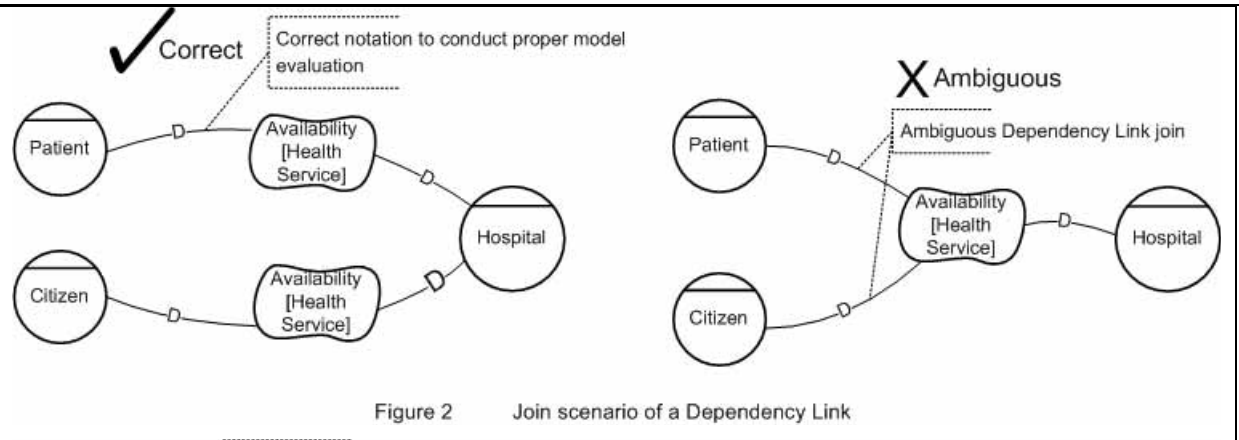


Figure 2 Join scenario of a Dependency Link

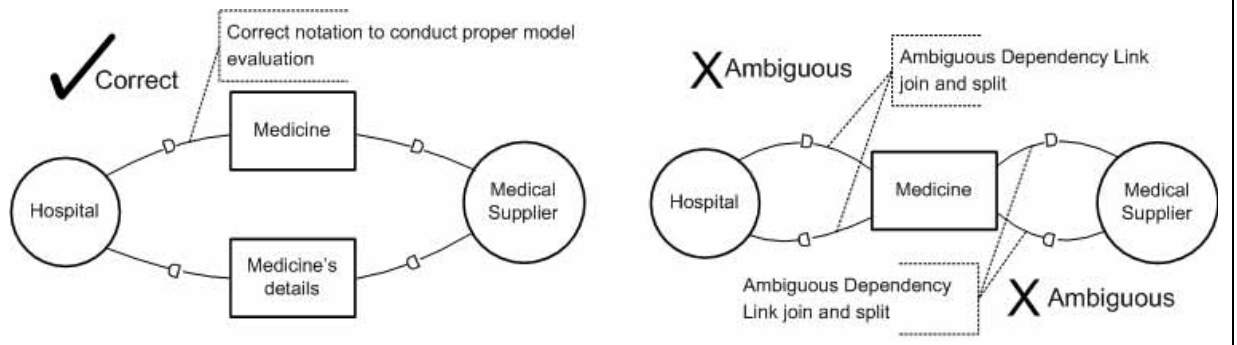
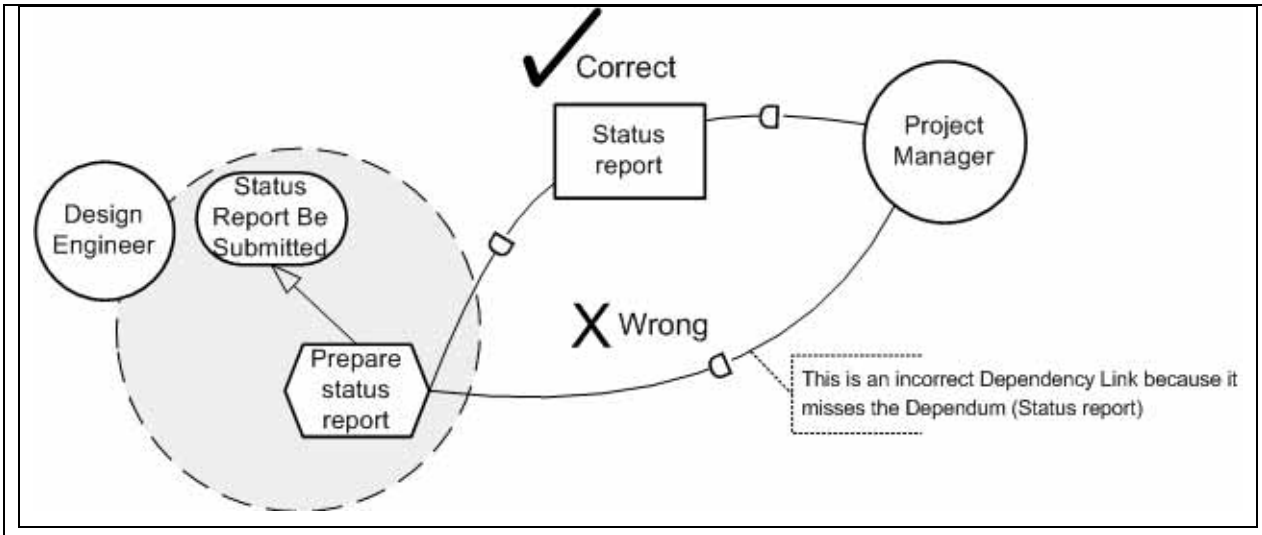


Figure 3 Join and split scenario of Dependency Links

4.1.3.11 Guideline (Beginner,Concept) Do not use a Dependency Link between two actors without showing the Dependum. Discuss

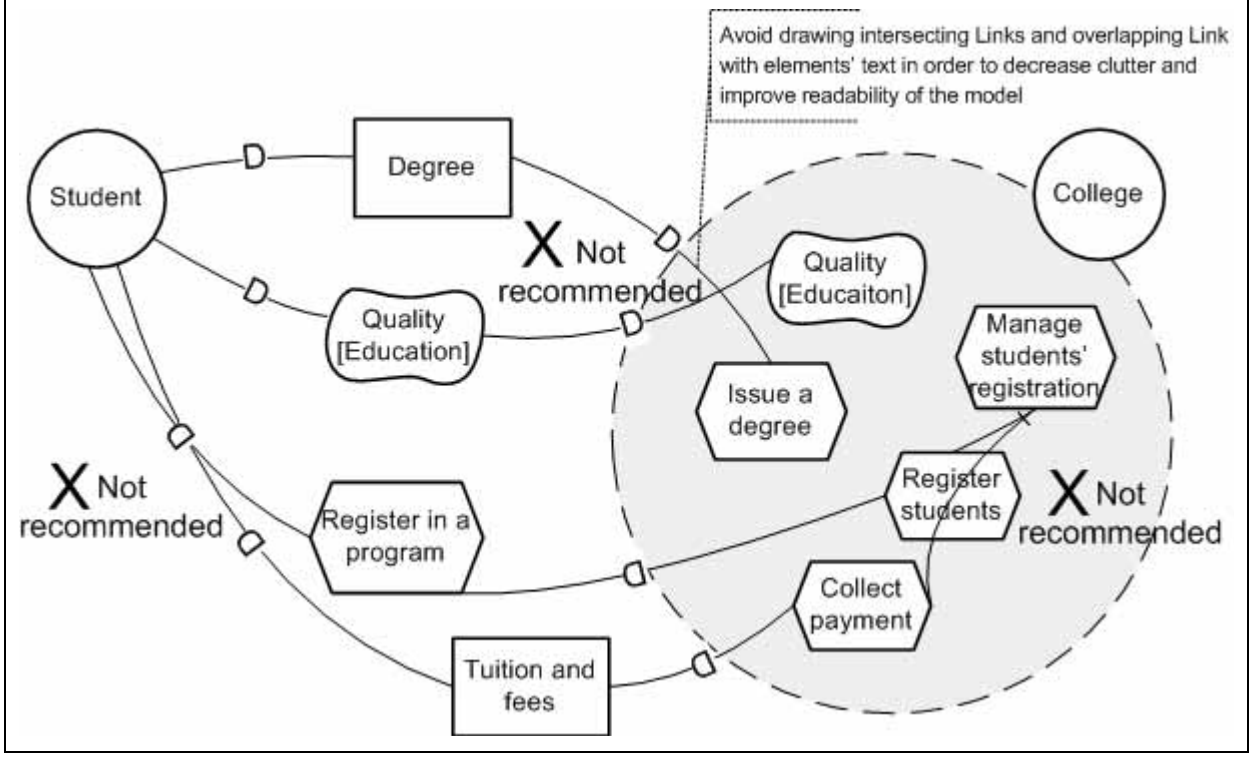
Discussion: The Dependency Link should contain a Dependum. Extending the Dependency Link from the Depender to the Dependee without showing the Dependum does not relay correct or complete information about the strategic dependency relationship.

For in-progress or unfinished models, the modeler might face a situation where the type and/or the naming of a Dependum has not been finalized. In this situation, using the non-conventional, direct link without a Dependum notation or a nameless Dependum might be temporarily acceptable, but with a corresponding comment in the Legend. The conventional Dependency Link, however, needs to be drawn properly by the time the model is finished. The reason behind taking such precaution is to avoid any confusion, misunderstanding, or misinterpretation by the other users of the model.



4.1.3.12 Guideline (Beginner,Layout) **Avoid or minimize drawing intersecting Links and overlapping Links with other Links and elements' text.** **Discuss**

Discussion: Overlapping Dependency Links between Actors or Decomposition and Contribution Links within Actors introduces complexity or clutter, which can worsen the readability and understandability of the model. Unless it is required to do so due to drawing space constraints, natural development of a complex model, or the model is very unambiguous, overlapping should be avoided or minimized.

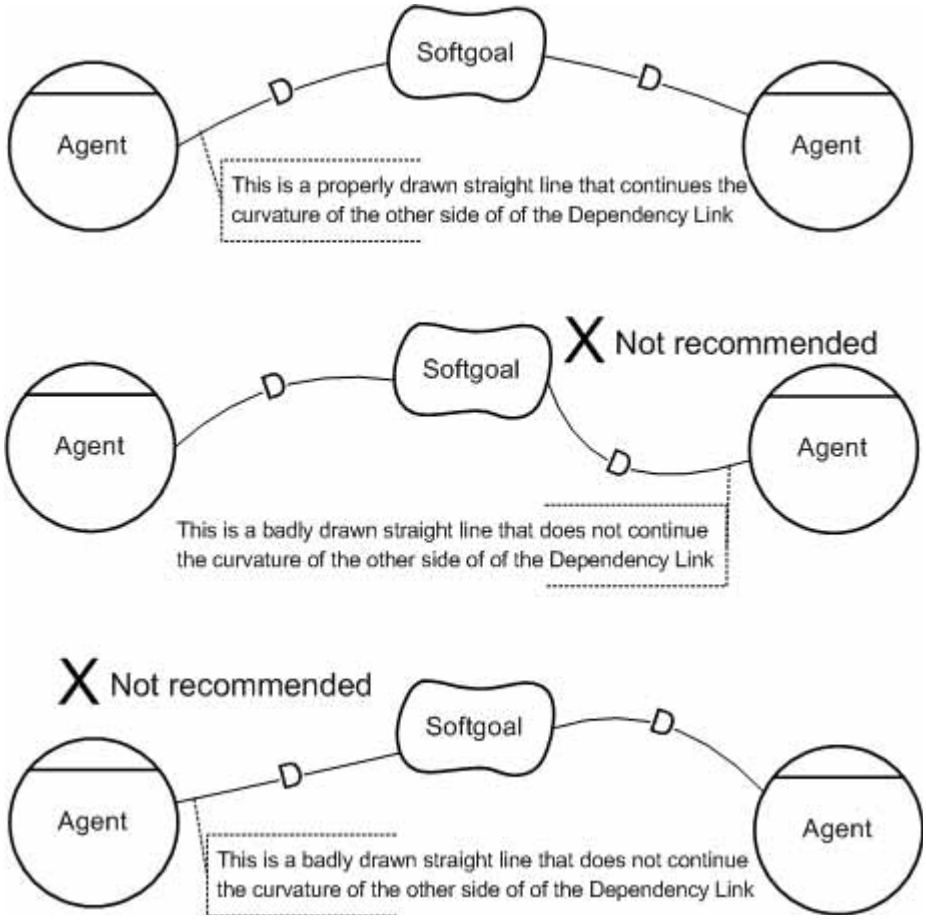


4.1.3.13 Guideline (Beginner,Layout) **Make both sides of a Dependency Link look like a single, continuous curve as it passes through the Dependum.** **Discuss**

Discussion: The outcome of making both sides of a Dependency Link look like a single, continuous

curvature as it passes through the Dependum is an enhanced presentation and improvement of the overall readability of the model. In some situations, such as when the model is very large and complex and the drawing space is limited, the modeler might need to have some flexibility by relaxing this guideline a bit. While it is allowed in such situations to break the rule, it is recommended to employ such guidelines as much as possible, especially in academic work and published articles.

Some tools or platforms such as the “istar_stencil” used in Microsoft Visio, require the modeler to manually adjust and arrange both sides of the Dependency Links to be drawn in a single, continuous curvature. Adjusting the snap and glue can help the user to adjust the link. Other tools such as “OME3”, takes care of this issue automatically.



4.1.3.14 Guideline (Beginner,Layout) Sperad the connection points of Dependency Links out on an Actor. Discuss

Discussion: Spread the connection points of Dependency Links out on an Actor as if each link will pass though the centre of the circle of the Actor. Clustering the Dependency Links connection points on certain spots on the Actors might make it difficult to visually distinguish between dependencies, and makes the model look cluttered. Some modeling tools such as Microsoft Visio,

where an i* stencil can be used, automatically snap the links to certain spots on the Actor. Users can configure the “Snap & Glue” feature in ‘Tools’ menu to help them manually adjust and position the links on the Actors.

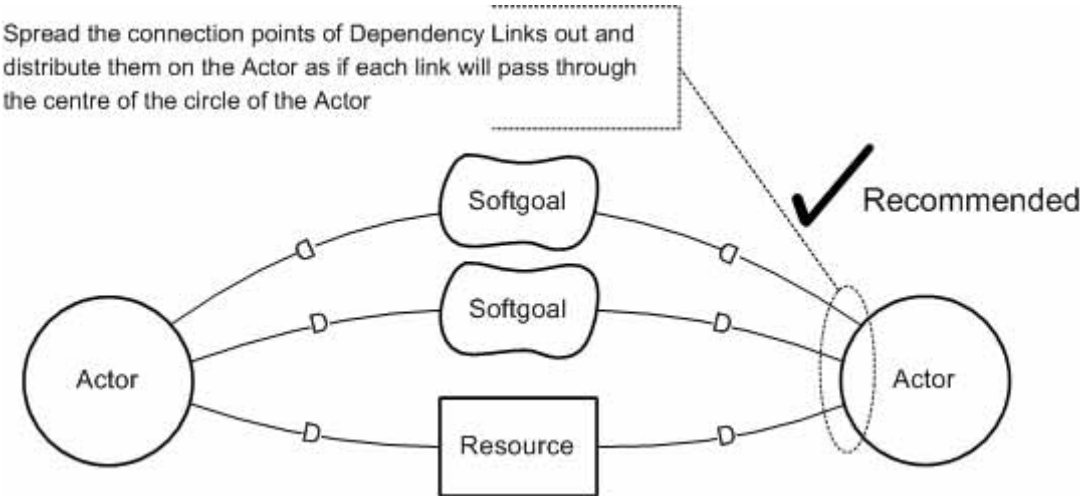


Figure 1 A recommended way of connecting Dependency Links on the Actor

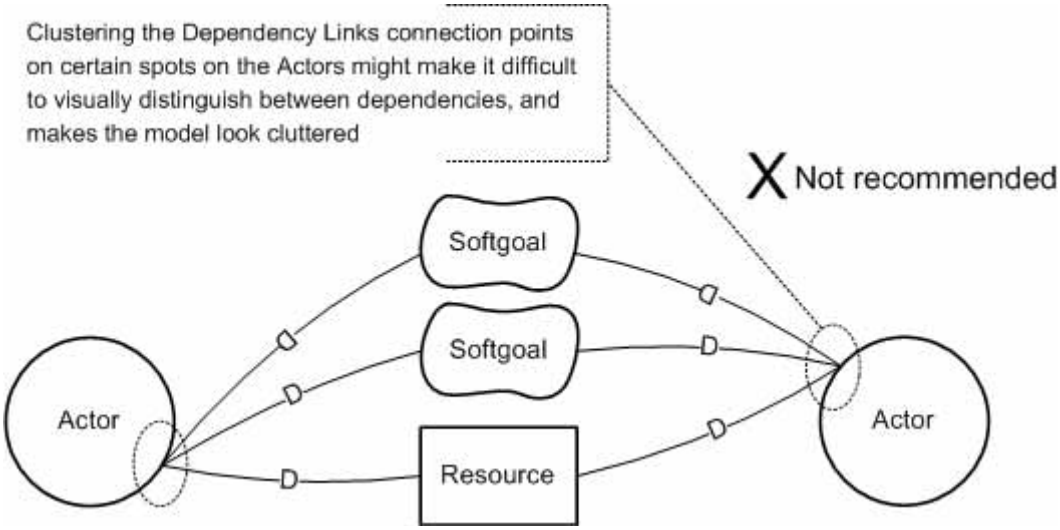


Figure 2 Clustering the connection on one spot is not recommended

4.1.3.15 Guideline (Beginner,Layout) **Keep elements horizontal. Do not tilt or twist them.** **Discuss**

Discussion: Maintain consistency in the orientation of the Dependums and internal elements to enhance the presentation and readability of the model. It is not recommended to tilt or twist the Dependums as this might impact the readability and presentation of the model.

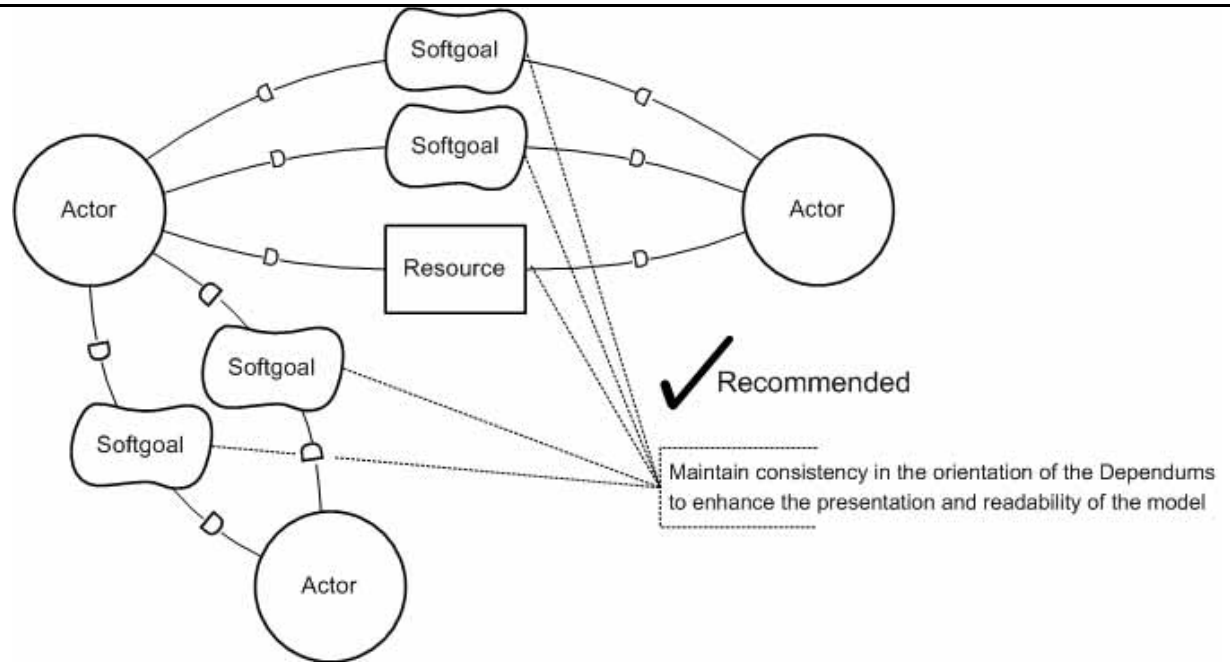


Figure 1 A recommended way of orienting the Dependums

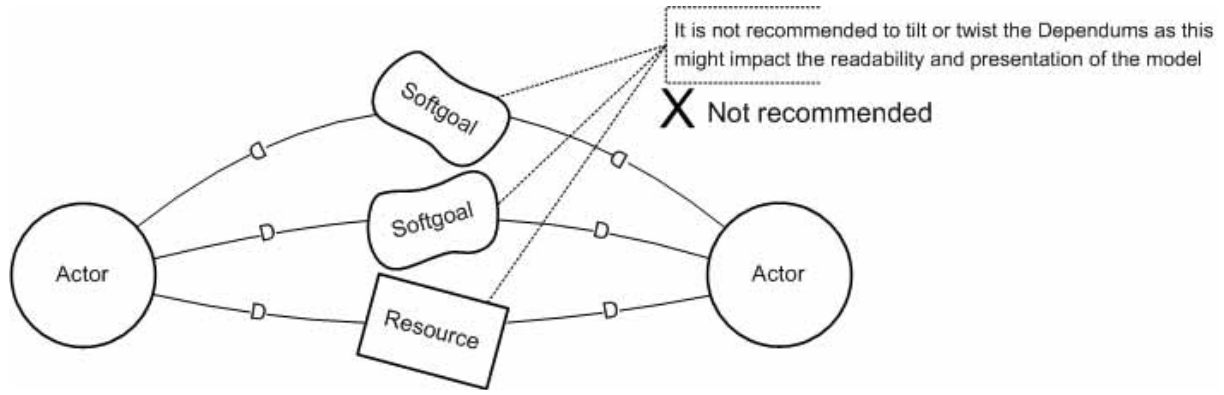


Figure 2 Tilting or twisting the Dependums is not recommended

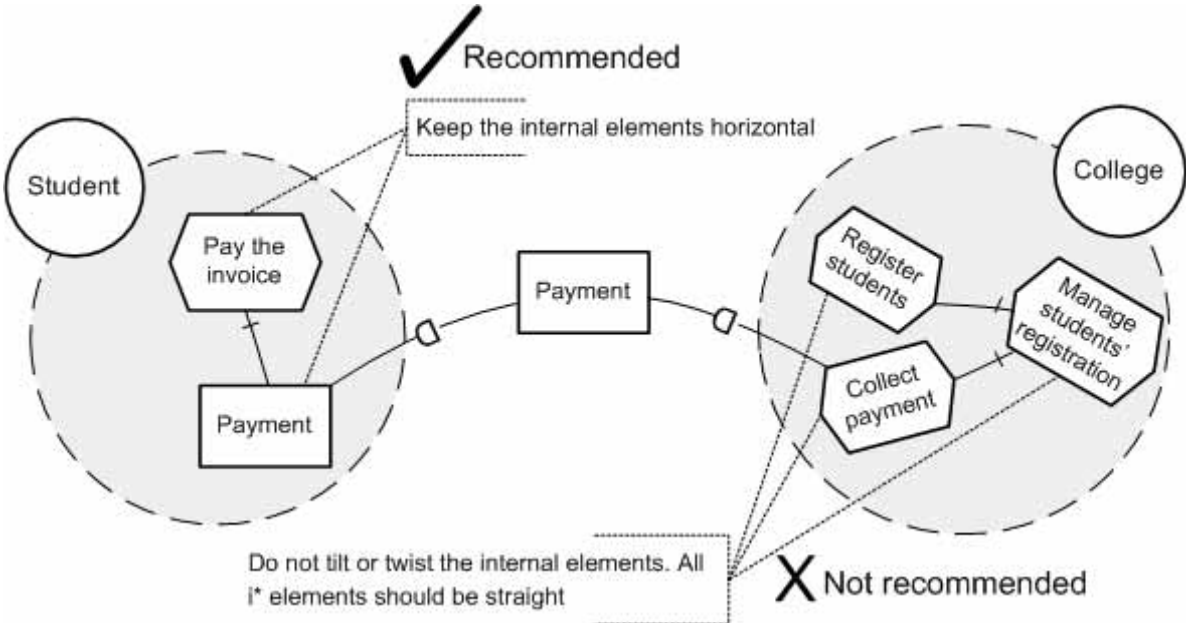


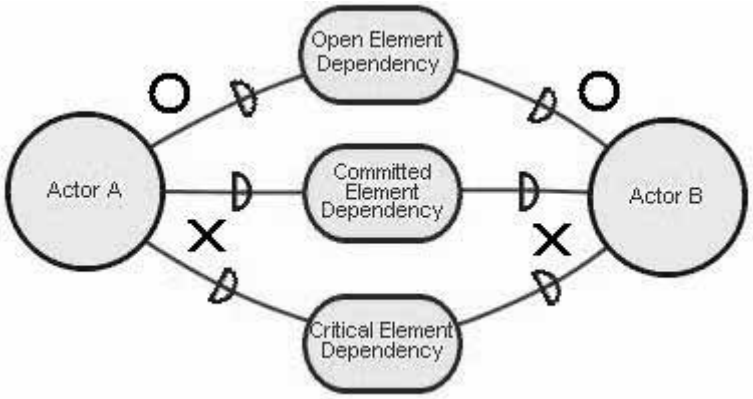
Figure 3 Tilting or twisting the internal elements is not recommended

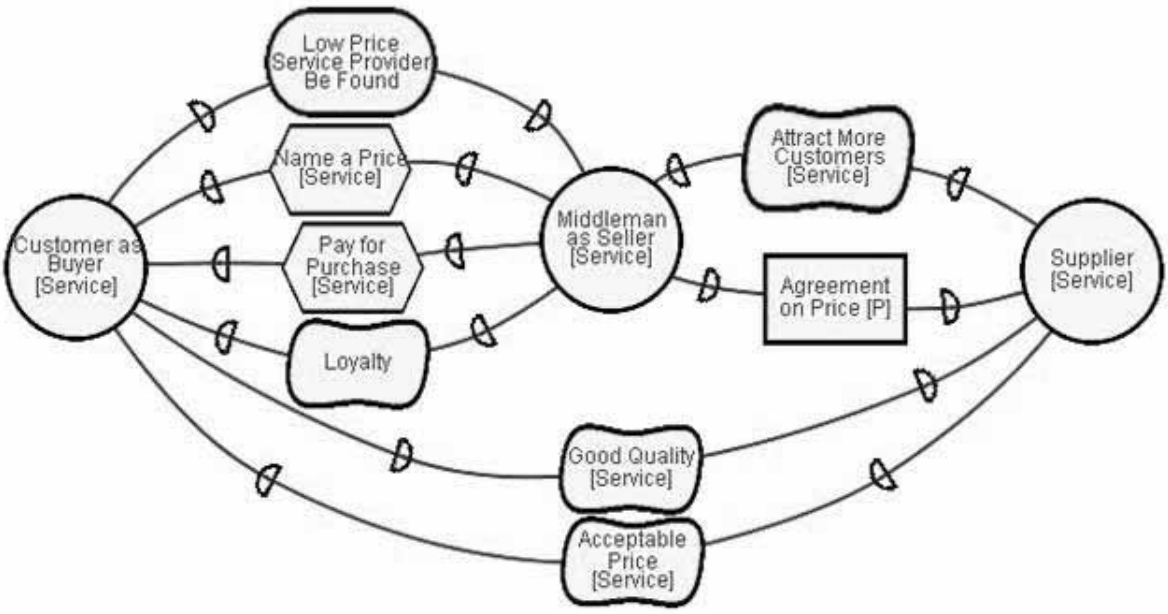
4.1.4 Vulnerability

Vulnerability is implied with Dependency Link(s). The dependency link represents that the depender, by depending on the actor who is the dependee, is able to achieve goals that it was not able to achieve before, or not as well, or not as quick. However this results in the depender becoming vulnerable to the intentions of the dependee. This vulnerability is implied because the dependee may fail to accomplish the specified element.

The model distinguishes three degrees of strength for the dependency according to the level of vulnerability. These types of dependencies apply independently on each side of a dependency. They are described in the following:

- Open dependency (uncommitted): Not obtaining the dependum would affect the depender to some extent but not seriously. This dependency strength is represented by including an "O" on the appropriate side of the link.
- Committed dependency: Not obtaining the dependum would cause some action for achieving a goal to fail in the depender.
- Critical dependency: Not obtaining the dependum would cause all actions to fail that the depender has planned to achieve a goal. This dependency strength is represented by including an "X" on the appropriate side of the link.





4.1.5 Strategic Dependency Example Model: Buyer Drive E-Commerce from Yu01

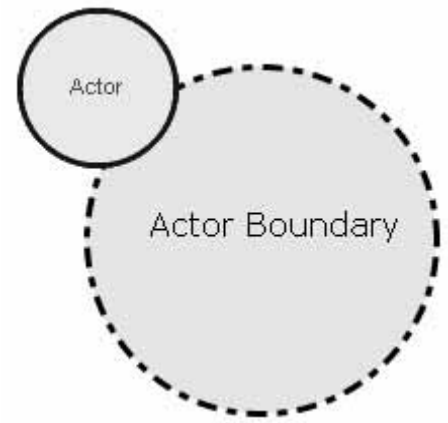
4.2 Strategic Rationale (SR) Model

The SR model is a graph, with several types of nodes and links that work together to provide a representational structure for expressing the rationales behind dependencies. The actors with the SD model are “opened up” to show their specific intentions. There are four types of nodes, based on the same distinctions made for dependum types in the SD model: goal, task, resource, and softgoal. There are three main classes of links internal to the i* actor: means-ends links, task decomposition links and contribution links.

SR diagrams open up Actors and show all the internal elements, including Goals, Softgoals, Tasks, and Resources that contribute to the analysis of alternatives and fulfillment of the dependencies. Goals and Softgoals can be attributed to not only human Actors, but also to non-human Actors (systems, machines, etc.) by the humans. For example, an email system (a non-human Actor) wants to be reliable, fast, and easy, because the humans who initiated these requirements or designed the system attribute these goals to the email system’s design.

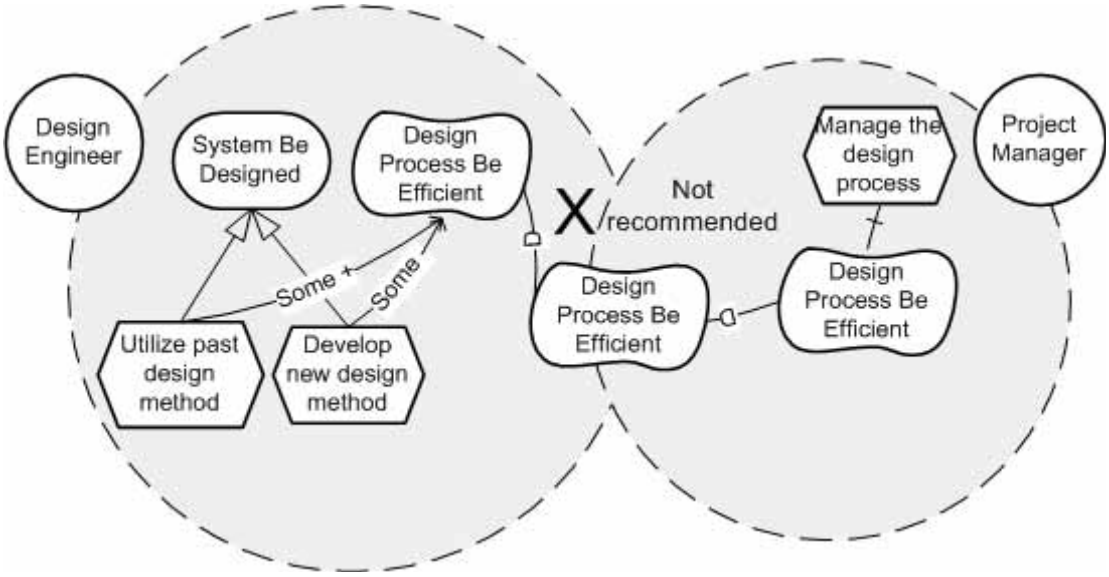
4.2.1 Boundary / Actor Boundary

Actor boundaries indicate intentional boundaries of a particular actor. All of the elements within a boundary for an actor are explicitly desired by that actor. In order to achieve these elements, often an actor must depend on the intentions of other actors, represented by dependency links across actor boundaries. In turn, an actor is depended upon to satisfy certain elements, represented by a dependency link in the opposite direction.



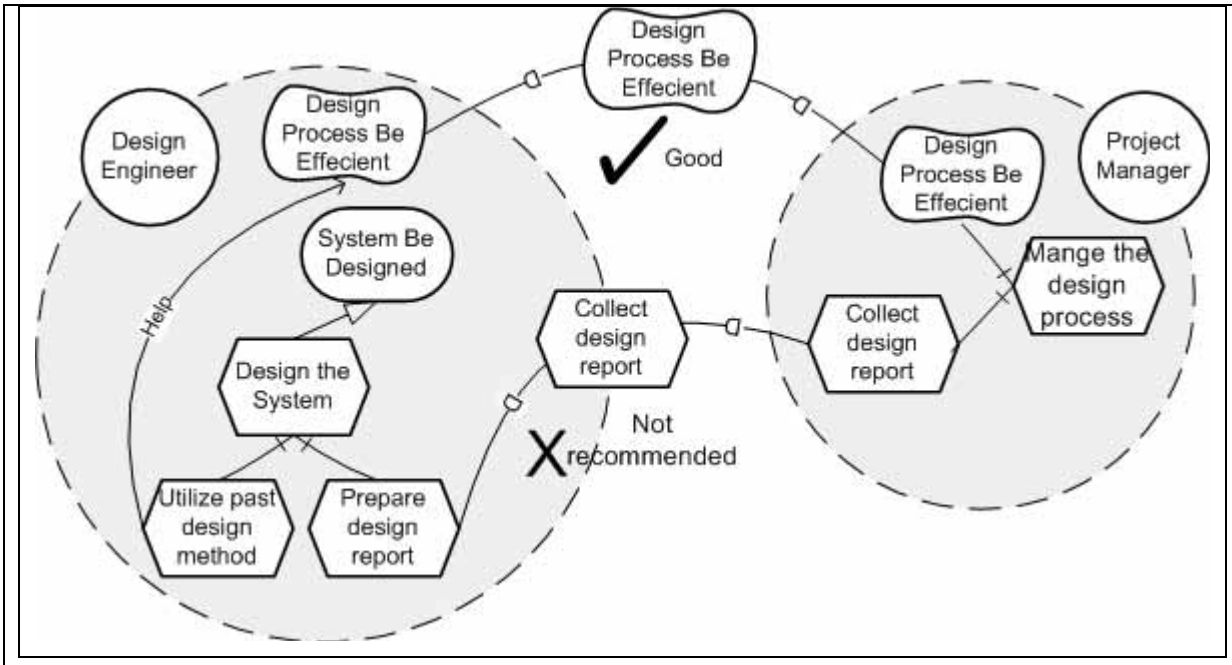
4.2.1.1 Guideline (Beginner,Layout) **Avoid or minimize overlapping boundaries of Actors where possible. Discuss**

Discussion: Where possible, it is desirable to draw the boundaries of actors in a clear manner without overlapping. Overlapping boundaries and Dependency Links makes models harder to understand, interpret, and communicate. It is recommended to leave enough canvas space as much as possible between actors to allow for the Dependency Links to be drawn clearly and eligibly.



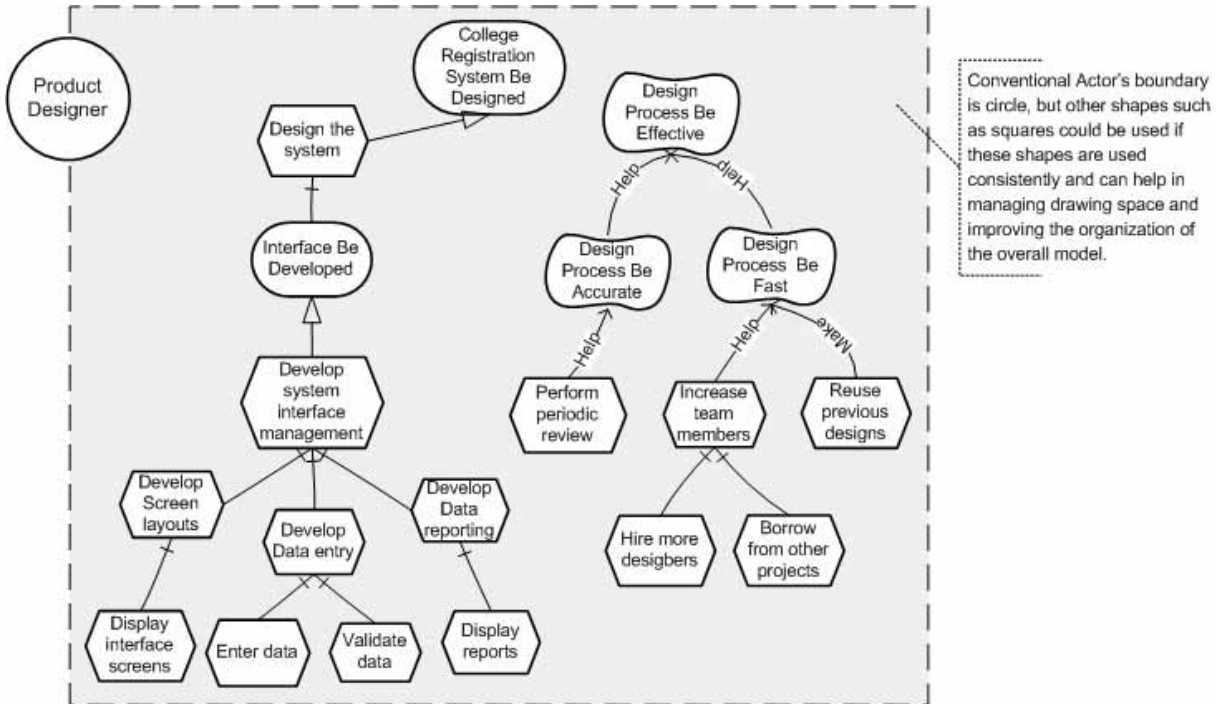
4.2.1.2 Guideline (Beginner,Layout) **Keep Dependency Links outside the boundaries of Actors to improve the readability of the models. Discuss**

Discussion: To make a Dependency Link clearly visible and recognizable, the Dependum as well as the “D” symbol should be drawn in space between and outside of the boundaries of Actors. Leave enough canvas space among actors to allow for the Dependency Links to be drawn clearly and legibly.



4.2.1.3 Guideline (Beginner,Layout) **Use the conventional Actors' boundaries (circles) unless other shapes such as rectangles can improve the overall layout. Discuss**

Discussion: The conventional Actors' boundaries are circles. In some instances other boundary shapes such as squares or rectangles can improve the management of the drawing space and therefore can enhance the readability and organization of the layout of the model. These non conventional shapes, however, need to be used consistently.



4.2.2 Elements/Nodes

The meanings of these elements are the generally the same as found in dependencies, with the exception that the satisfaction of elements may be
accomplished internally.

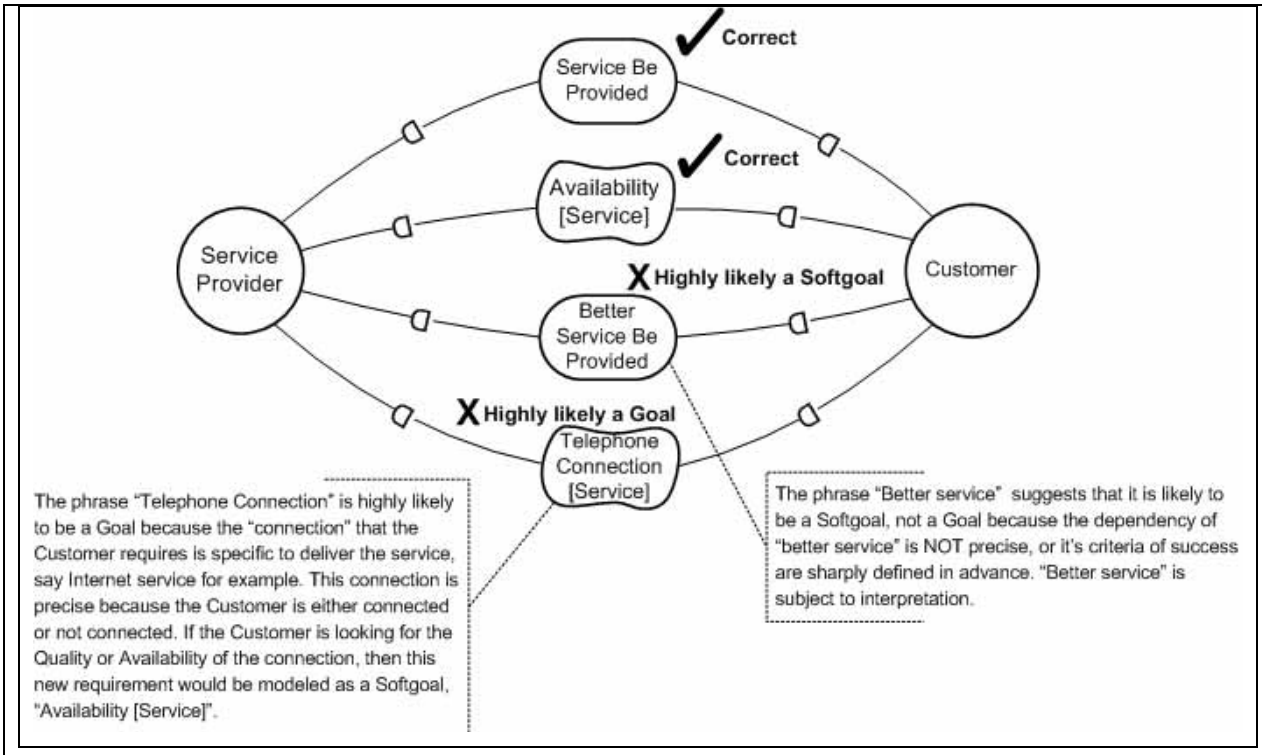
4.2.2.1 Goals (Hard Goals)

Represents and intentional desire of an actor, the specifics of how the goal is to be satisfied is not described by the goal. This can be described through task decomposition.

4.2.2.1.1 Guideline (Beginner,Concept) Use a Softgoal for quality criterion and use a (hard) goal for a sharply defined objective. Discuss

Discussion: There are goals (hard goals) and softgoals. Examples of goals are: Product Be Designed, Service Be Provided, and Team Be Hired. Examples of softgoals are: Design Process Be Efficient, Low Product Cost, and Availability Service. Not every goal is a softgoal. Use softgoals when modeling quality attributes or non-functional requirements (NFRs) or use softgoals when stakeholders' goals are not precise or their criteria of success are not sharply defined in advance. On the other hand, a Goal is precise and its end state or outcome is clearly specified. Goals are part of the functional requirements.

As shown in the illustration, the phrase "Better service" suggests that it is likely to be a Softgoal, not a Goal. "Better service" is NOT precise, or it's criteria of success are sharply defined in advance. "Better service" is subject to interpretation. Conversely, "Telephone Connection" is highly likely to be a Goal because the Customer is either connected or not connected. If the Customer is concerned about the Quality or Availability of the connection, then this new requirement would be modeled as a Softgoal, "Availability [Service]".



4.2.2.1.2 Guideline (Beginner,Concept) Do not confuse Softgoal with optional, less important Goals. The criticality, or importance, or priority of a goal is orthogonal to the distinction between a hard goal and a Softgoal. Discuss

Discussion: The distinction between a (hard) Goal and a Softgoal is not in the degree of importance. To model something as a Softgoal is not to say that it is less important, or that it is optional. For example, Performance[online update] might be modeled as a critical Softgoal. In i*, dependencies may be assigned one of three levels of "strength" - Critical, Committed, or Open. The strength attribute is separate from the distinction between a Softgoal and a hard Goal

4.2.2.1.3 Guideline (Beginner,Concept) To indicate that a Goal can be achieved by performing several sub-tasks, model the decomposition by introducing a Task. Discuss

Discussion: Use a single Means-Ends Link to connect a Task to a Goal, then, decompose this Task into sub Tasks using Decomposition Links. In the example, the Task "Provide health service" is decomposed into the all the sub Tasks below it. Also, these sub Tasks can be decomposed further, if required based on the nature and context of the analysis, into more sub Goals, Softgoals, Tasks, and Resources.

Figure 1 depicts a correct scenario of a Goal decomposition by introducing a Task. Figures 2 and 3 illustrate incorrect scenarios where the main goal is incorrectly and directly decomposed using Decomposition and Contribution Links.

Note: See the guideline, "Don't mix Goals and Tasks in the Means-Ends links" for further discussion

about Goal decomposition and a note about the difference between i* and Tropos in decomposing Goals into sub-goals.

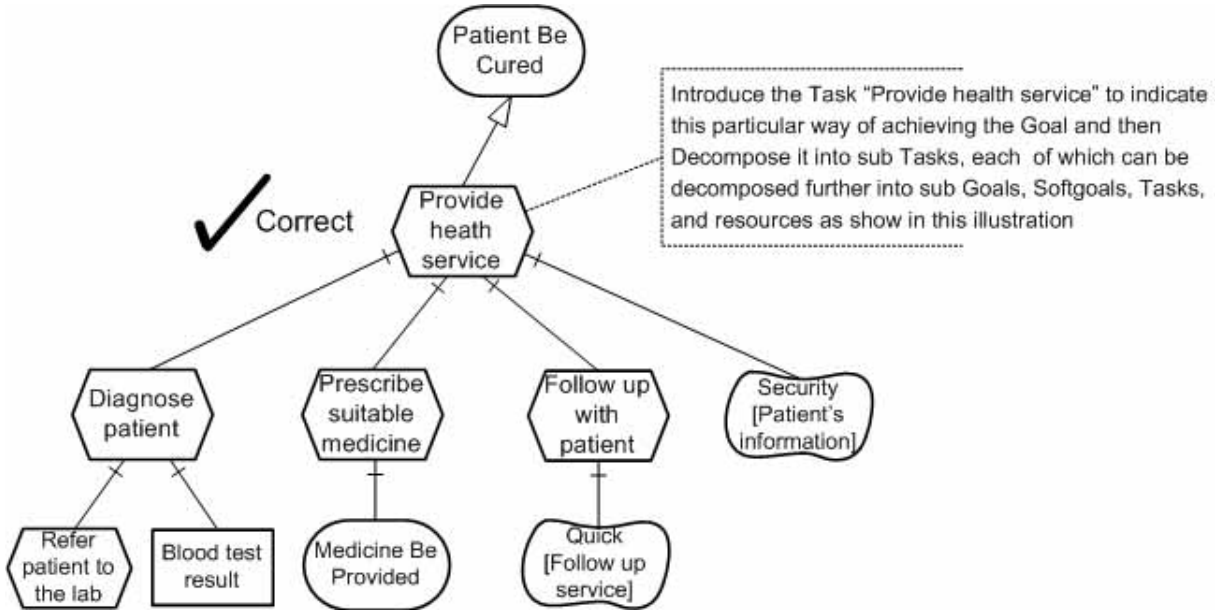


Figure 1 A correct scenario of a Goal decomposition by introducing a Task

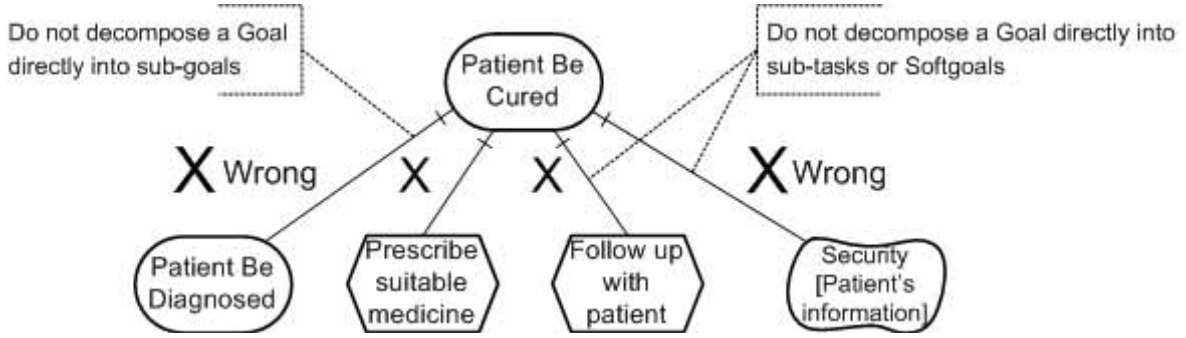


Figure 2 A wrong scenario where the goal is incorrectly and directly decomposed using Decomposition Links

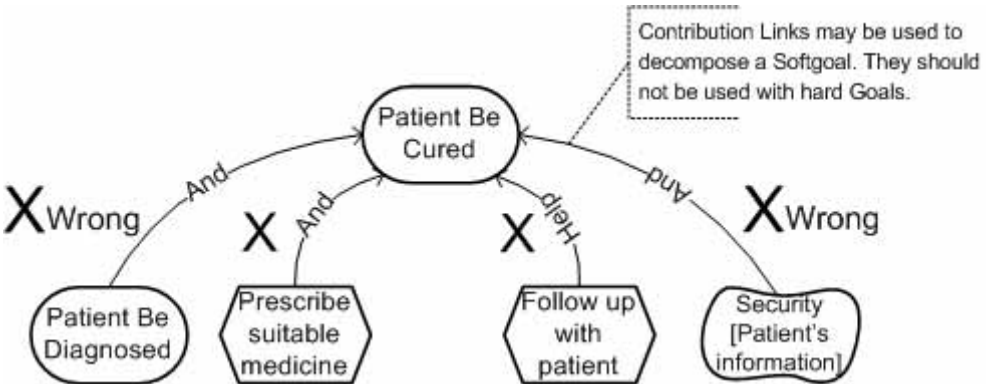
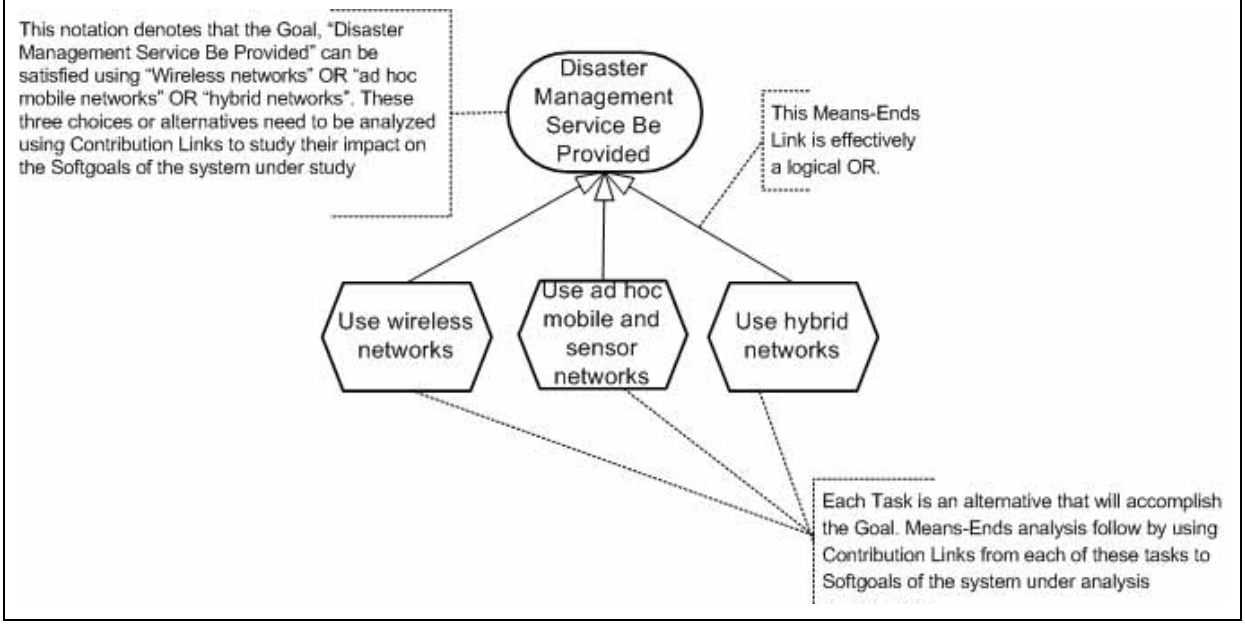


Figure 3 A wrong scenario where the goal is incorrectly and directly decomposed using Contribution Links

4.2.2.1.4 Guideline (Beginner,Concept) Use multiple Means-End Links from Tasks to a Goal to indicate alternatives. Discuss

Discussion: Using this notation gives the modeler a way to analysis the alternatives of satisfying a Goal. The impact of each alternative on the Softgoals or NFR of the system is analyzed using Contribution Links such as: Make, Some+, Help, Break, Some-, Hurt, and Unknown.



4.2.2.1.5 Guideline (Beginner,Concept) Don't mix Goals and Tasks in the Means-Ends links. Discuss

Discussion: Goals are different than Tasks, and therefore can't be mixed in Means-Ends Links. A Goal is an Actor's End desire, not a Means to achieve this End desire (Goal). In i* framework, An End desire (Goal) is achieved via a Task, which is a Means to achieve the End desire or a Goal. See Section 1.4.2.2 "Elements/Nodes" for more explanation about Goals and Tasks. Figure 1 depicts an example where the Means-Ends Links is allowed and where it is not. See the guideline in section 1.4.2.2.1 "Goal (Hard Goals)" for an example of using the Means-Ends Link and Task notations to indirectly decompose Goals to sub-goals.

Note: The i* Means-Ends Link notation is not to be confused with a similar notation used by Tropos methodology to denote an OR decomposition of Goals. Figure 2 (troposproject.org) depicts Tropos notation for both AND and OR decomposition notation. It can be noticed that Tropos allows direct decomposition of Goals to sub-goals. On the other hand, i* decomposes Goals into sub-goals indirectly using the Task notation as shown in Section 1.2 "Basic i* notation" and discussed in Section 4.2.4 "Decomposition Links". The i* notation of Goal decomposition encourages the modeler to consider distinctions among Goals, Softgoals, Tasks, and Resources are each step of decomposition.

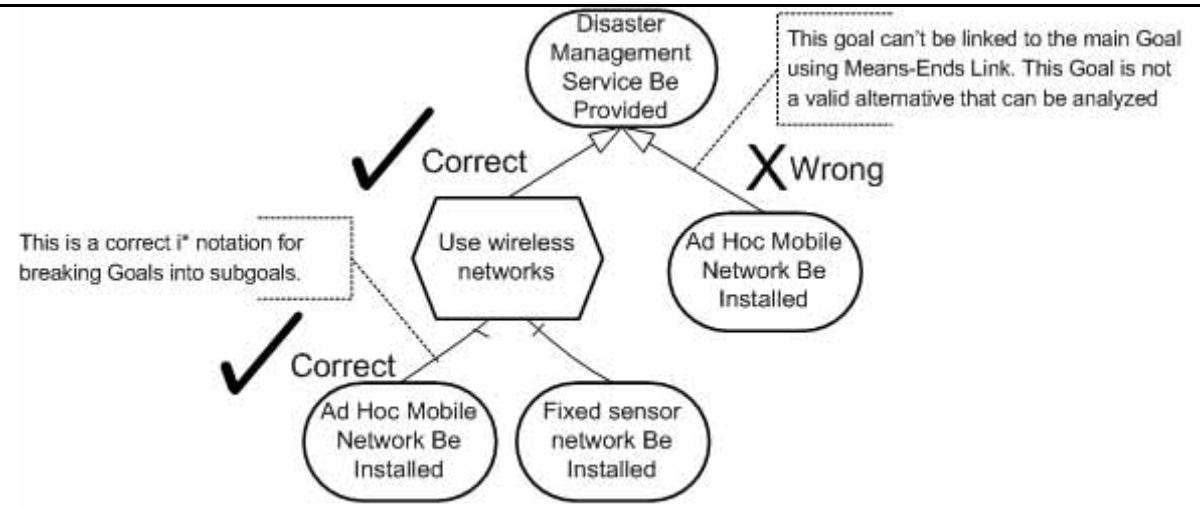
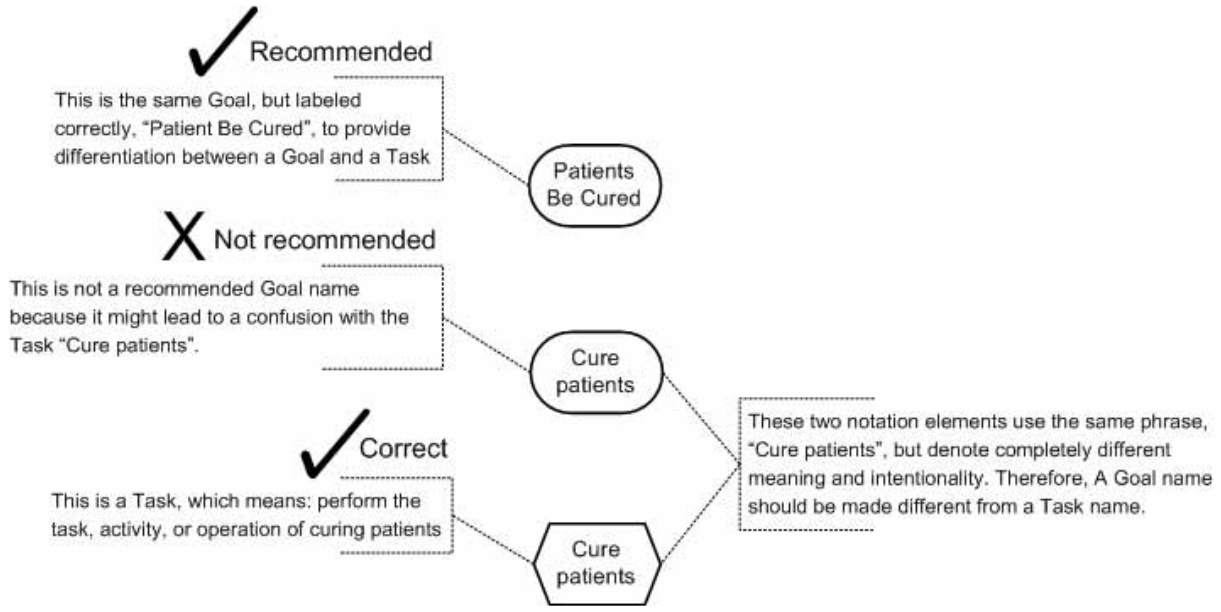


Figure 1 Correct use of Means-Ends Link notation with Goals in i*

Figure 2 AND and OR Goal decomposition notation in *Tropos*

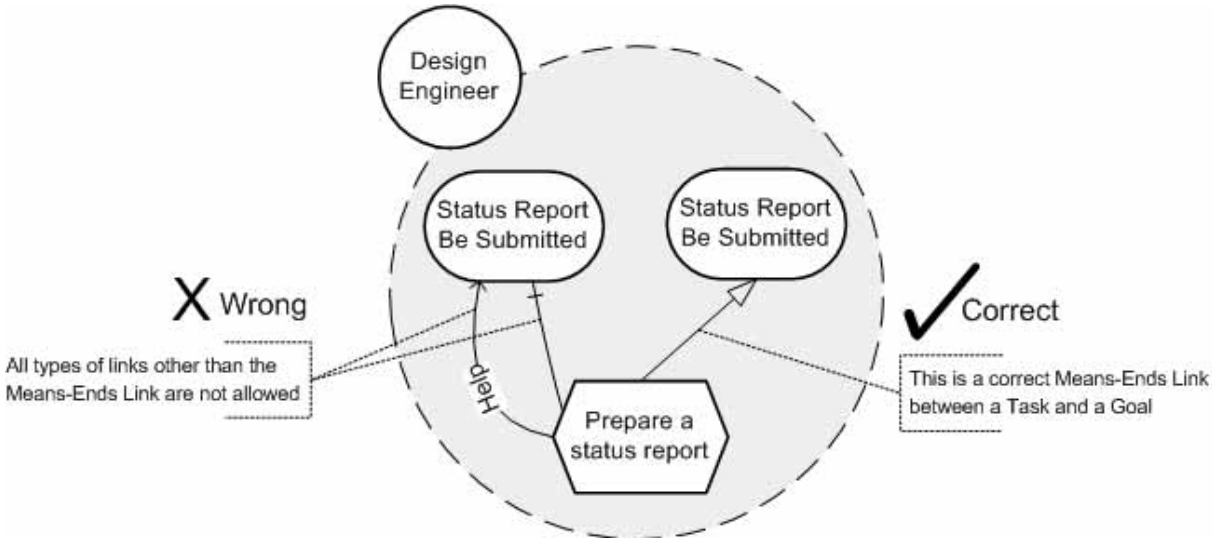
4.2.2.1.6 Guideline (Beginner,Naming) **Use precise language to name a Goal or a Task. Discuss**

Discussion: Goals have the naming convention of "Goal Be Achieved" such as: Product Be Developed, Infrastructure Be Installed, and System Be Tested. Goals should not to be confused with the Tasks, which start with verbs such as: Collect the payment, Process the request, and Test the module. Sometimes, deviation from this Goal naming convention could be allowed, however, if the naming convention seems a bit unnatural.



4.2.2.1.7 Guideline (Beginner,Concept) **A Goal can only be decomposed using Means-Ends Links. Discuss**

Discussion: The Means-End Link is a type of a relationship that indicates an End (Goal) and it's Means (Task), or how to achieve the Goal. Using a Means-End Link indicates that the Means fulfills the goal. This indicates that an alternative which would satisfy the goal has been discovered. Further alternatives should often be sought. Contribution Links such as Help, Some+, etc. are used with Softgoals, not with Goals. Also, Decomposition Links are used to decompose a Task, not a Goal. Refer to Guideline 4.2.3.1 for additional discussion on using Means-Ends with Goals.

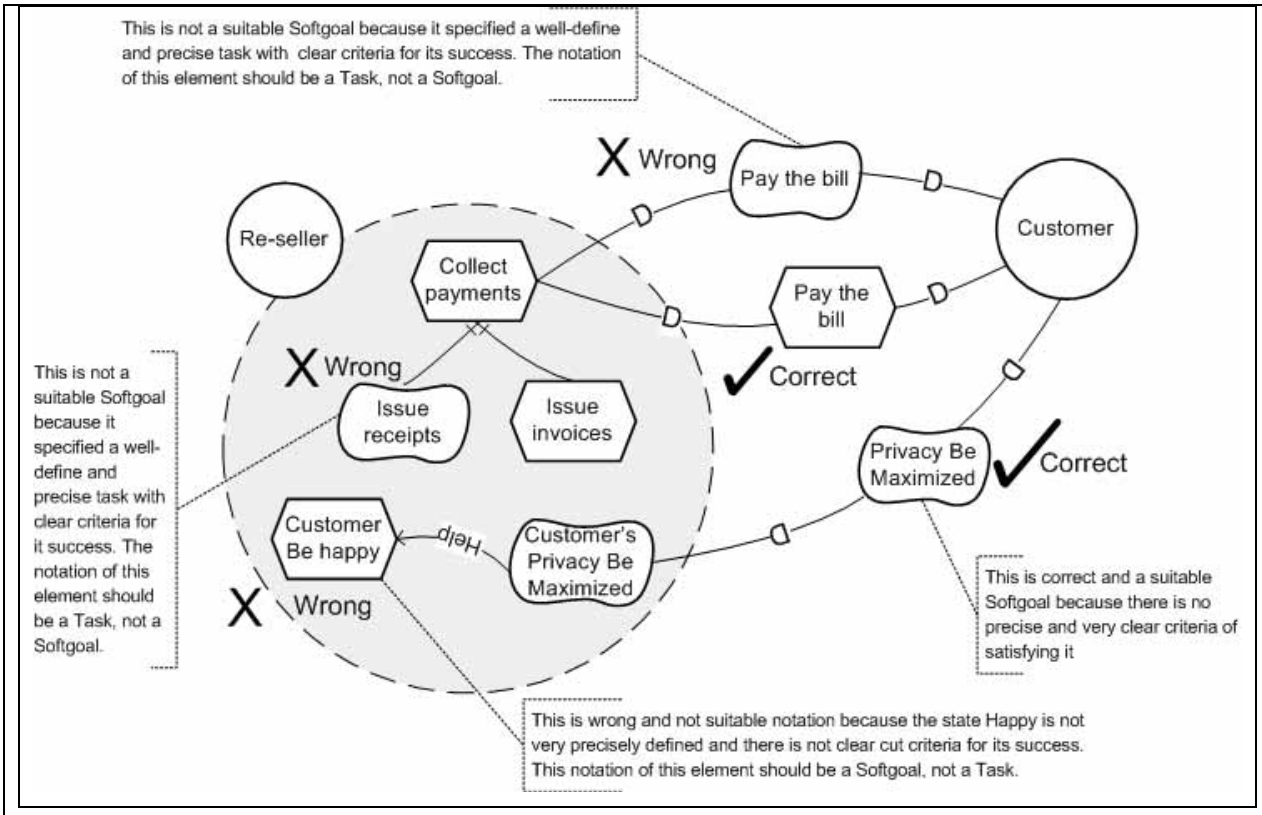


4.2.2.2 Softgoals

Softgoals are similar to (hard) goals except that the criteria for the goal's satisfaction are not clear-cut, it is judged to be sufficiently satisfied from the point of view of the actor. The means to satisfy such goals are described via contribution links from other elements. The notion of softgoal satisfaction is described by the term satisficed meaning sufficiently satisfied. The converse is still described as denied.

4.2.2.2.1 Guideline (Beginner,Concept) **Do not confuse between a Softgoal and a Task. Discuss**

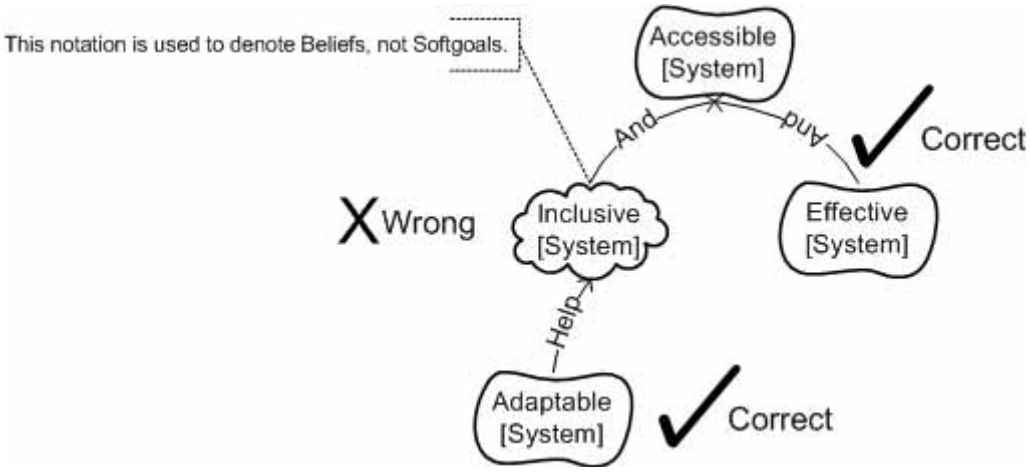
Discussion: Softgoals are used when modeling quality attributes or non-functional requirements (NFRs). Also Softgoals are used when stakeholders' goals are not precise, may change, or their criteria of success are not sharply defined in advance. Tasks are used when an actor (Depender) depends on another actor (Dependee) through the strategic Dependency Link, to accomplish a task, which has a clear idea of achieving it. The Dependee, however, still has freedom to carry out the activity within some constraints. A Task is usually is described by decomposing it into sub elements, which could be another task, softgoal, goal, or resource. Also, Tasks don't always have to be involved in a dependency. An actor can have internal tasks, which are not involved in a dependency.



4.2.2.2.2 Guideline (Beginner,Notation) **Use the proper i* Softgoal notation. Discuss**

Discussion: The graphical notation for a Softgoal in i* is the "squished" rectangle with rounded corner shape. Do not use the "cloud" shape which is used to denote a Belief in i*.

Note: The "cloud shape is used to denote a Softgoal in the NFR Framework and in Tropos.



4.2.2.2.3 Guideline (Intermediate,Concept & Evaluation) **Softgoals and Goals should be decomposed. Discuss**

Discussion: One of the objectives of creating i* models is to show how Goals and Softgoals can be

achieved through operationalization, or though more concrete actions and design decisions included in the model. This process of finding alternatives which satisfy goals and sofgoals is called “operationalizing” or decomposing elements. Therefore, modelers are encouraged to try not to leave Softgoals and Goals as leafs elements, or elements which do not have any incoming link. Figure 1 depicts one scenario where a Softgoal is left as a leaf. Figure 2 depicts a general example of when a Softgoal is considered a leaf element or a non-leaf element. Figure 4 depicts an example of Goal leaf and non-leaf elements. In cases such as this, the model is incomplete, not all of the requirements or (soft)goals contained in the model can be achieved through the alternatives included in the model. If this Softgoal leaf element is left unconnected with other model elements through Contribution Links and/or Dependency Links, it will be difficult to know whether this Softgoal (requirement) is satisficed. It may be allowable to leave Softgoals as leafs in initial, incomplete models, but later, effort should be made to find ways to achieve all Softgoals. Figure 2 illustrates connecting such a Softgoal leaf to other model elements using incoming Contribution Link(s) or Dependency Link(s) in order to show how Softgoals are achieved.

Refining and decomposing Goals and Softgoals provide the advantage of developing more complete and accurate models. More accurate models, however, might lead to more complex models, which impacts the time, effort, and cost of the design process. Therefore, the trade-off between these different aspects should be taken into consideration. It may be advisable to leave softgoals and goals as leaf elements if the model is very large, but this incompleteness should be noted and justified.

Refer to Section 4.2.6 ‘Leaf Elements’ for definition of leaf element.

Refer to Section 4.2.8 ‘Operationalizations and Refinement of Goals and Softgoals’ for more information about refining and decomposing Goals and Softgoals.

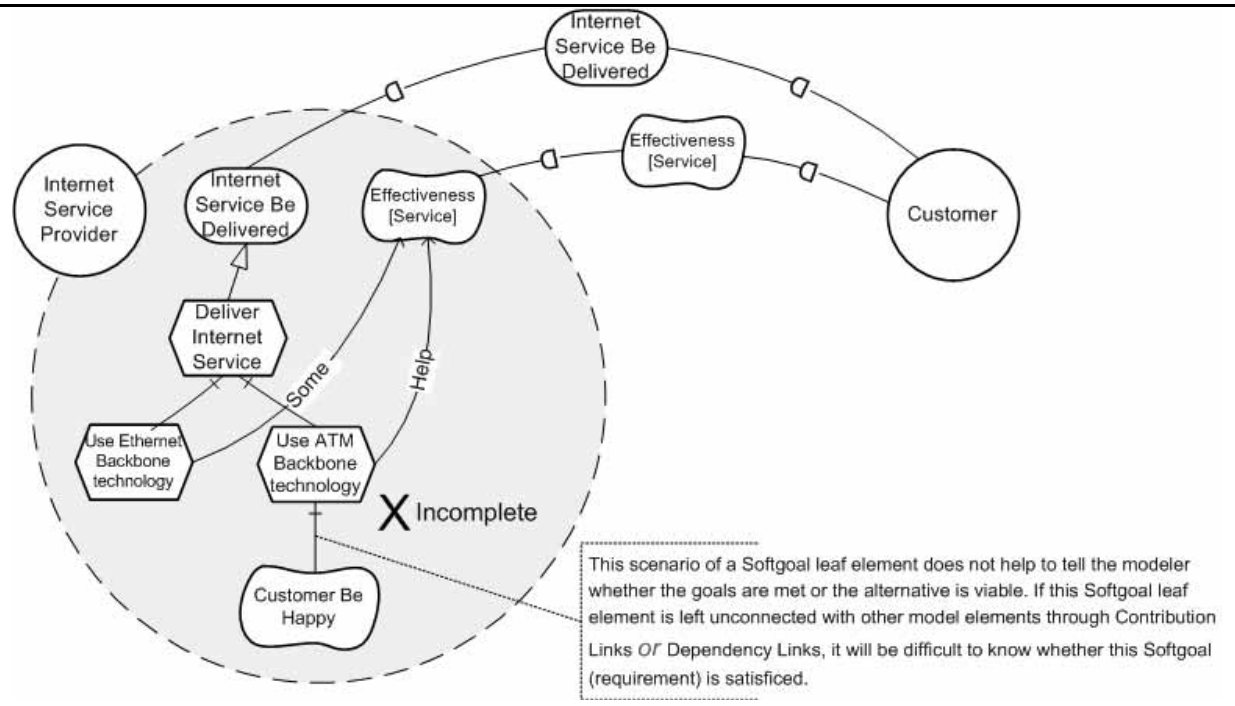


Figure 1 One possible scenario where a Softgoal is left as a model leaf

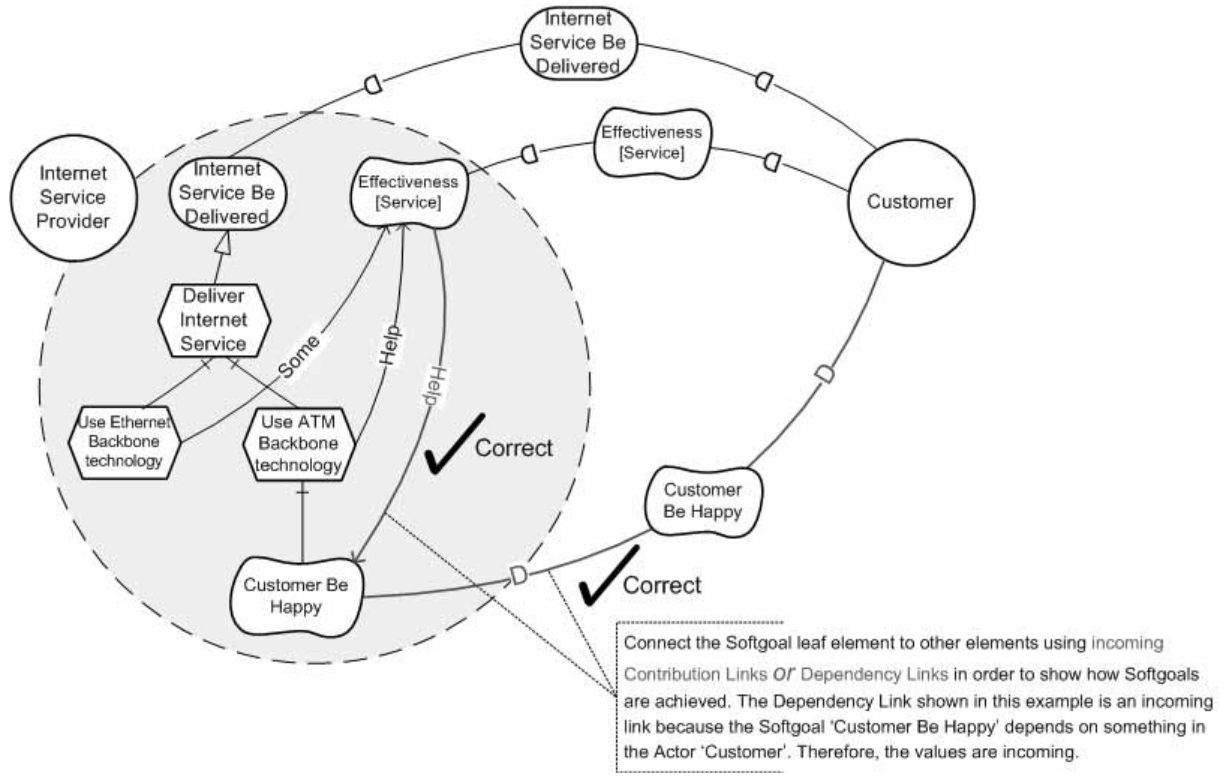


Figure 2 Two ways to avoid leaving a Softgoal as a model leaf

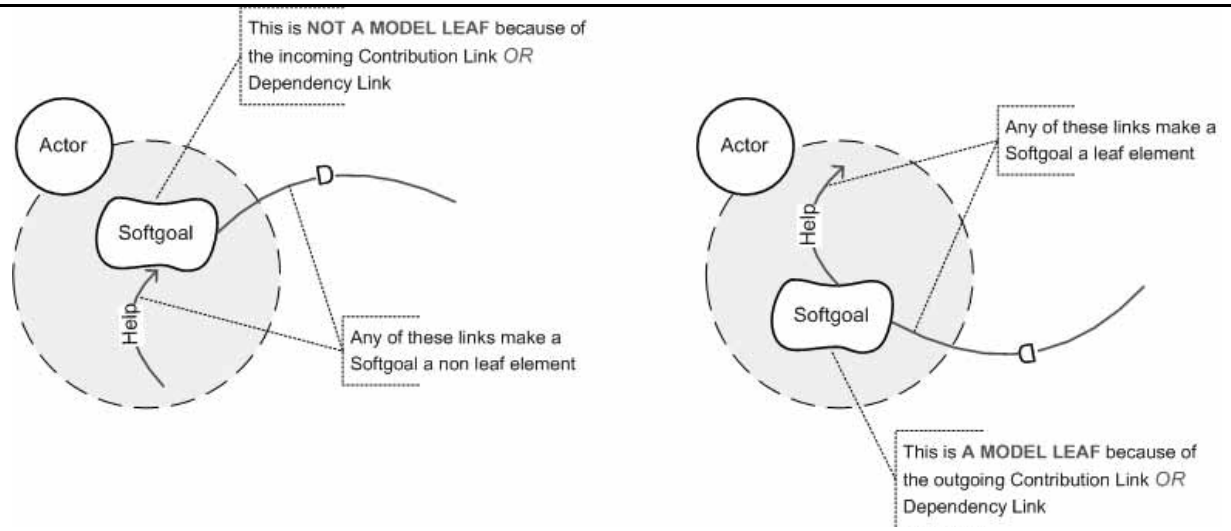


Figure 3 A general example of when a Softgoal is considered a leaf element or a non-leaf element

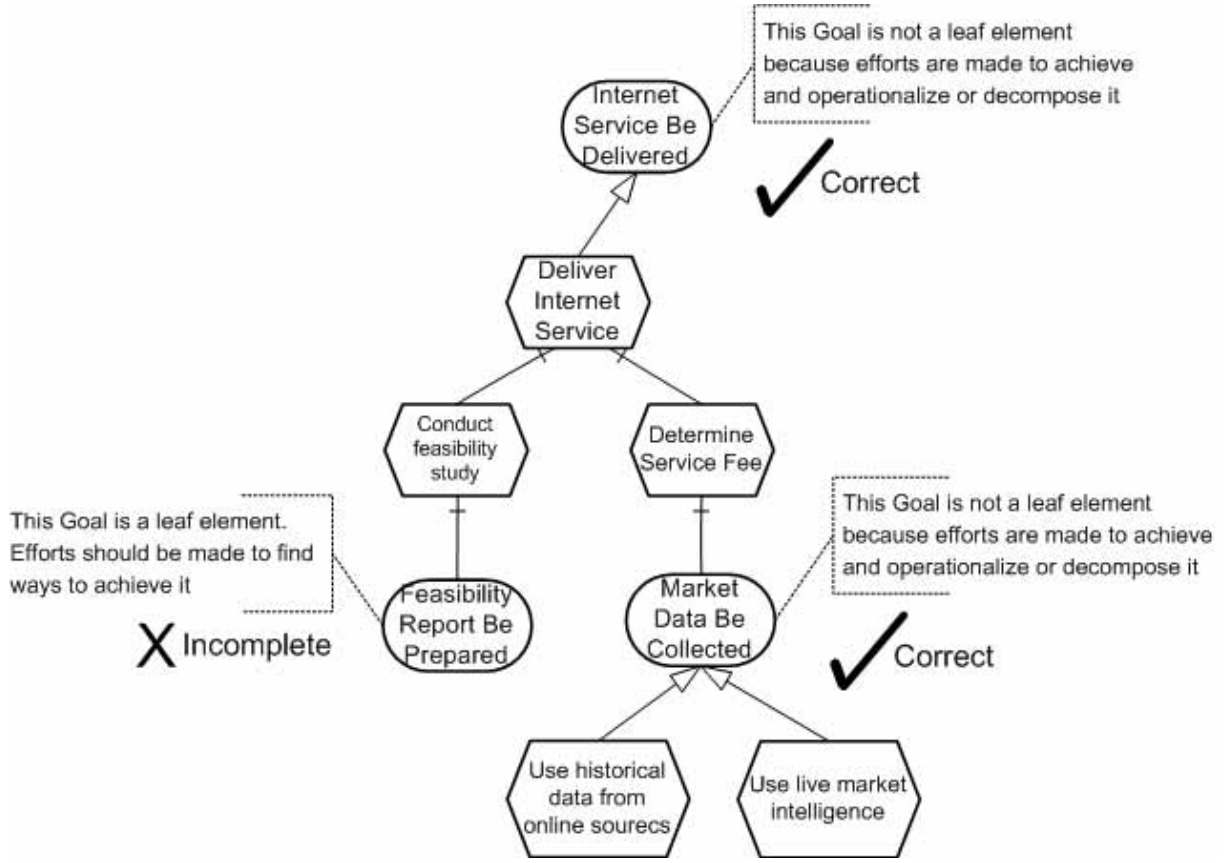


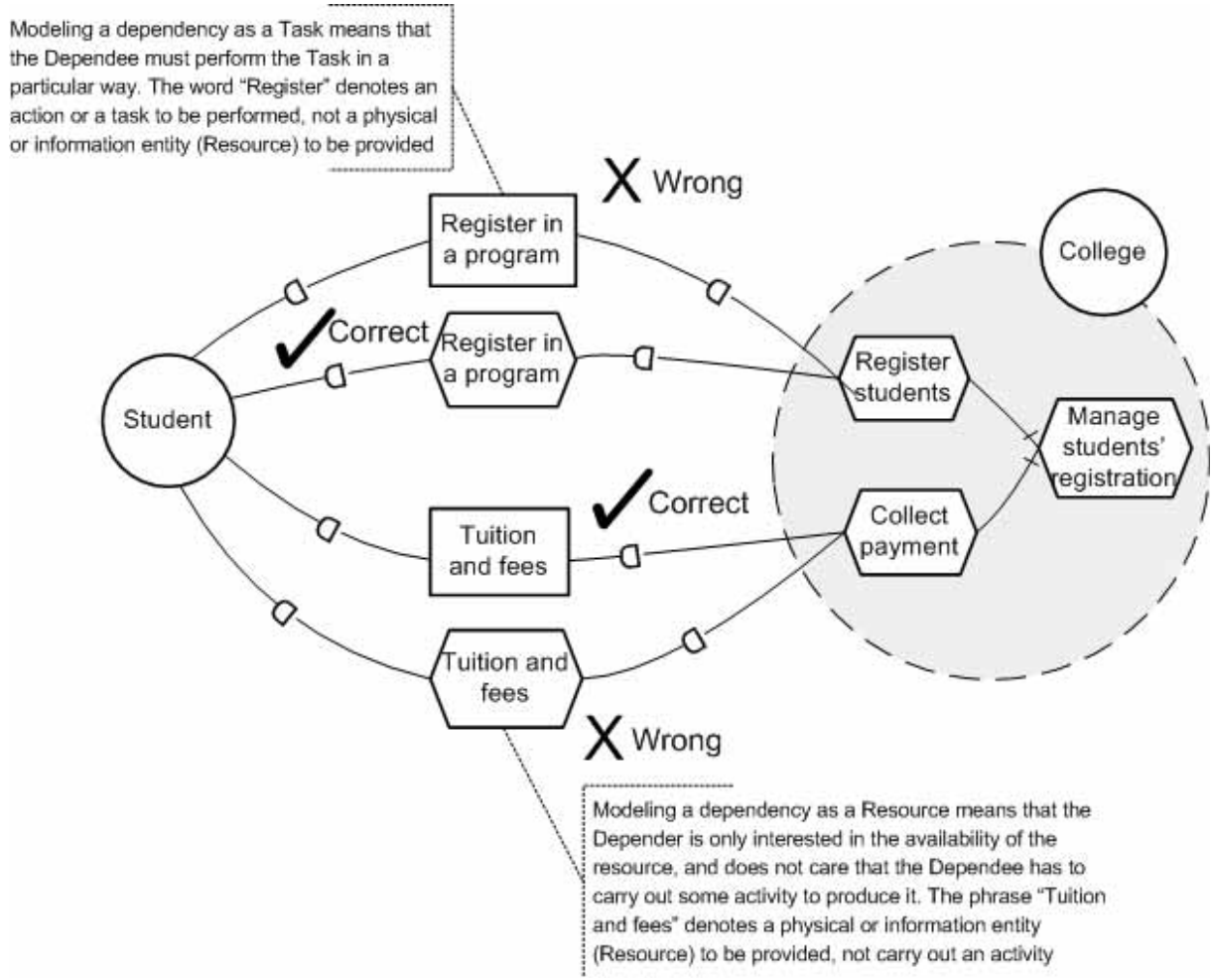
Figure 4 Scenario where Goals are leaf and non-leaf elements

4.2.2.3 Tasks

The actor wants to accomplish some specific task, performed in a particular way. A description of the specifics of the task may be described by decomposing the task into further sub-elements.

4.2.2.3.1 Guideline (Beginner,Concept) Do not confuse between a Task and a Resource. Discuss

Discussion: In the Task Dependum, the Depender depends on the Dependee to perform a Task for which there is clear criteria of achieving the Task. A Task is an activity, which needs to be performed by the Dependee, who still has freedom to carry out the activity within some constraints. Modeling a dependency as a Task means that the Dependee must perform the Task in a particular way. On the other hand, a Resource Dependum denotes that the Depender depends on the Dependee for the availability of a resource such a physical resource or information. Modeling a dependency as a Resource means that the Depender is only interested in the availability of the resource, and does not care that the Dependee has to carry out some activity to produce it

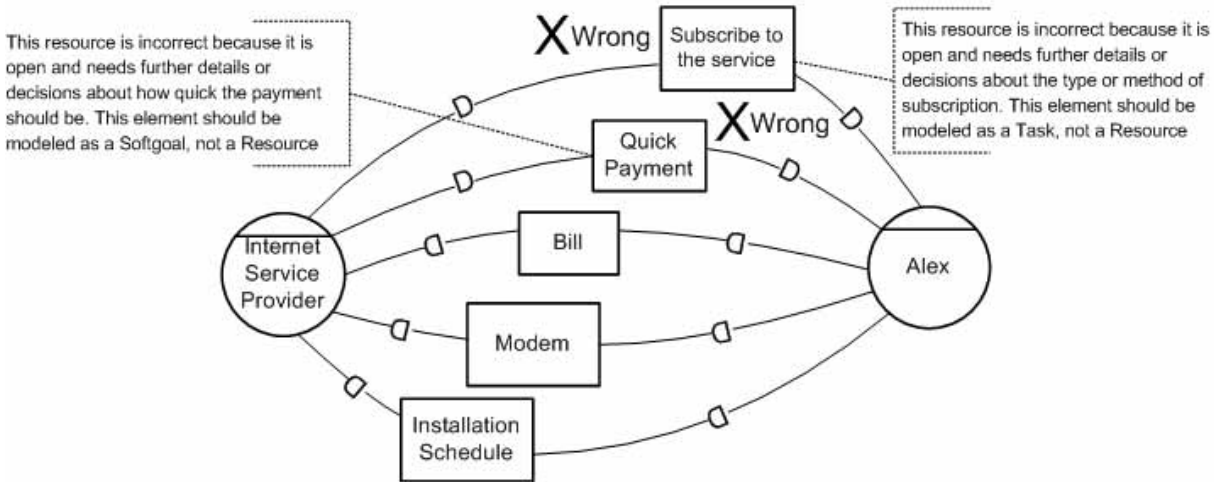


4.2.2.4 Resources

The actor desires the provision of some entity, physical or informational. This type of elements assumes there are no open issues or questions concerning how the entity will be produced or provided.

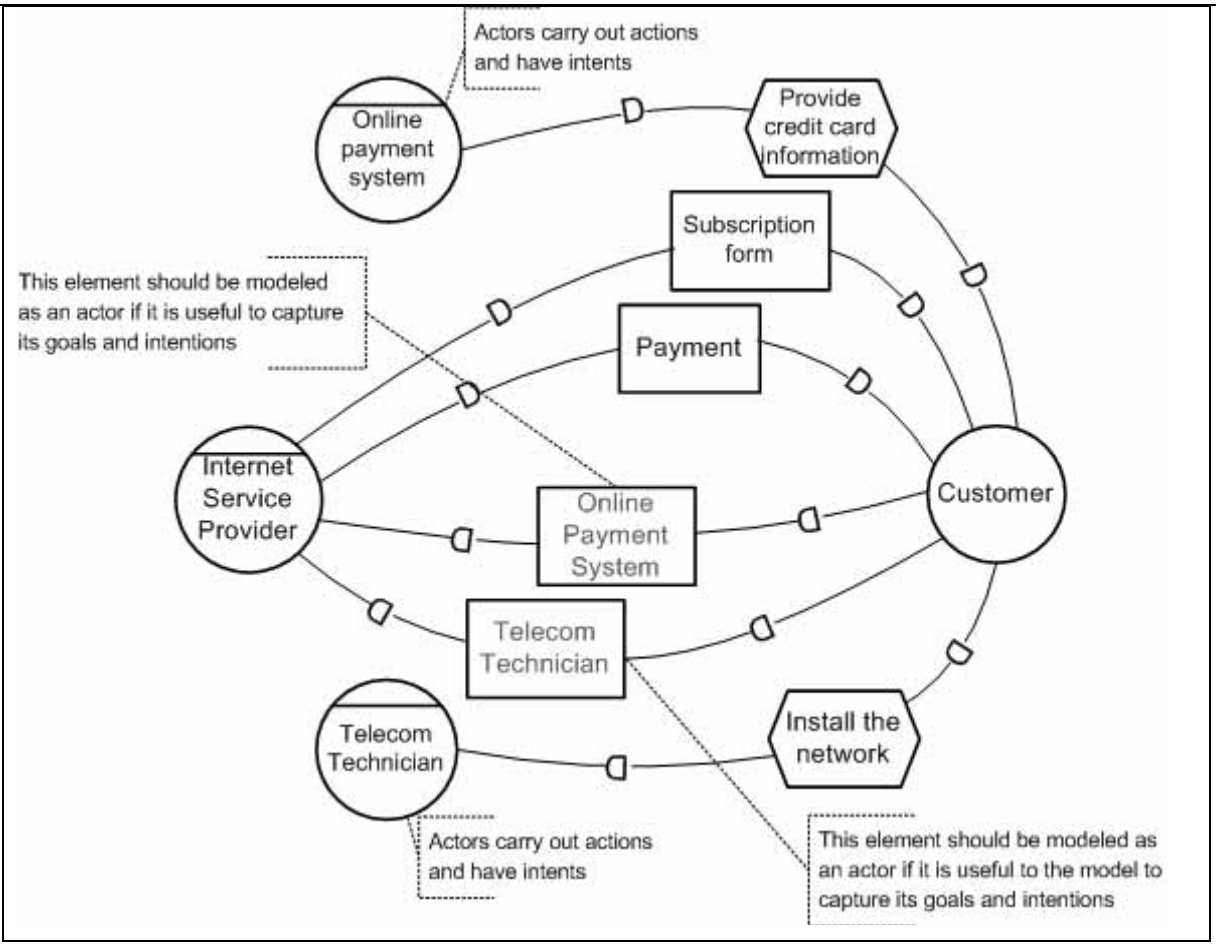
4.2.2.4.1 Guideline (Beginner,Concept) **Use a Resource when the Actor asks for the provision of a clearly defined and concrete resource, which can be physical or information entity. Discuss**

Discussion: The Resource element should be clear, concrete, and well defined. Resources that are open in nature and prompts for additional actions or decisions to be made for their provision suggest using alternative elements such as Task, Goals, or Softgoals. Please refer to Section 4.1.3.3 "Resource Dependency" for discussion about the Resource entity.



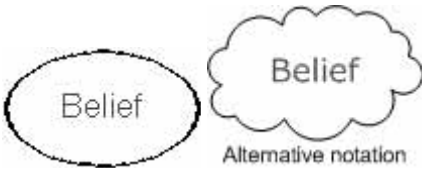
4.2.2.4.2 Guideline (Beginner,Concept) **Model a human or system as a resource only if you want to ignore their goals and intentions. Discuss**

Discussion: An Actor is an entity that performs an activity or an action. If the Depender depends on an entity to perform a task, action, or activity to fulfill, satisfy, accomplish, or achieve a goal, then that entity is modeled as an Actor, not a Resource. In the supplied example, "Online Payment System" and "Telecom Technician" are not passive deliverables, but active elements that performs action within the overall model. Therefore, they need to be modeled as some type of actors. Moreover, the modeler decides if it is useful to model the Goals and intentions of the "Telecom technician", for example, then it should be an Actor, otherwise, in a simple model, it could be a resource.



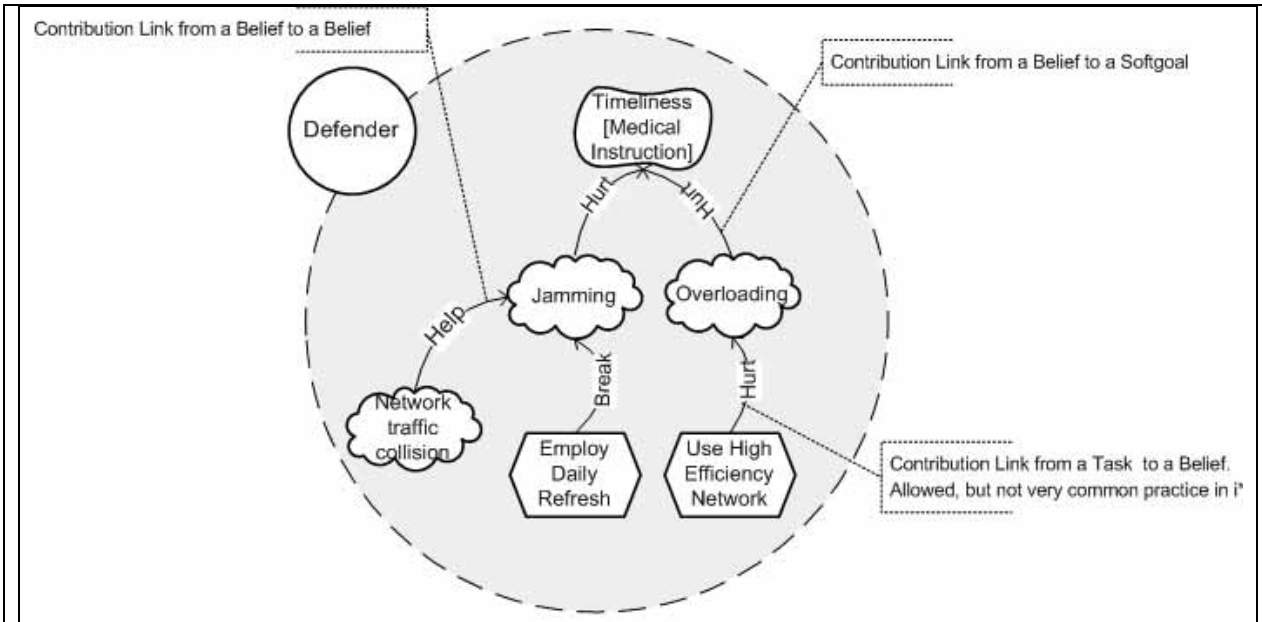
4.2.2.5 Beliefs

A belief is a condition about the world that the actor holds to be true. The actual degree of truth (As indicated by evaluation labels) is influenced by contributions from other beliefs. A belief is distinct from a goal in that the actor has no explicit desire to make the specified condition become true.



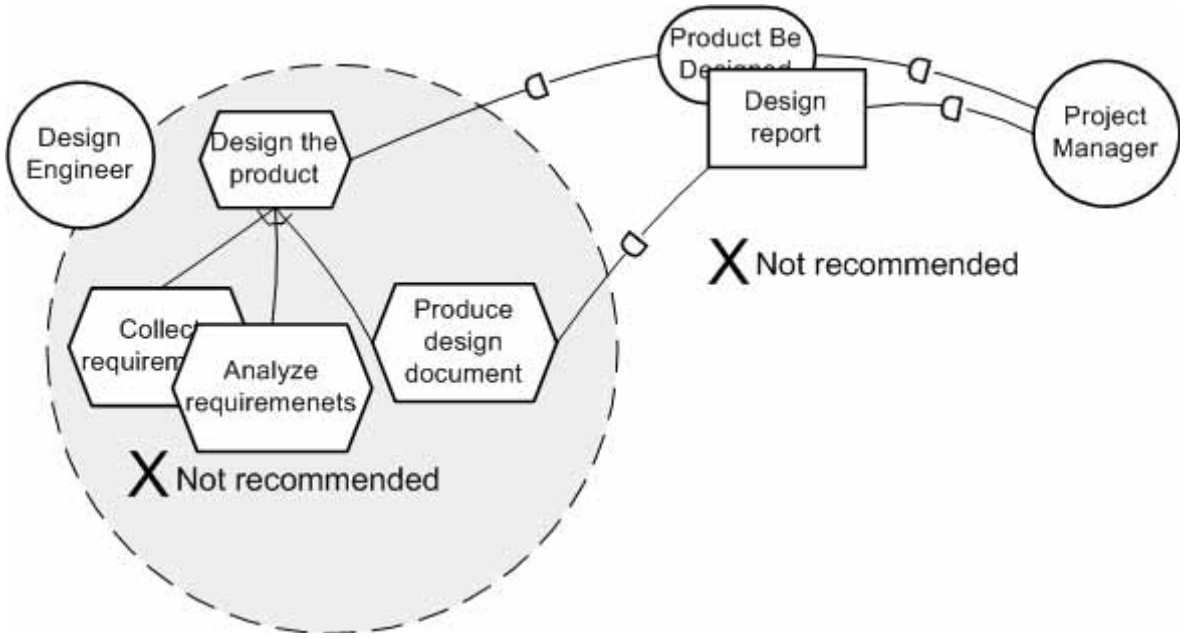
4.2.2.5.1 Guideline (Beginner,Concept) Describe the effect of Beliefs on Softgoals using Contribution Links. Discuss

Discussion: Please refer to section 4.2.2.5 Beliefs for definitions of the element Belief. Also, please refer to section 4.2.5 Contribution Links for definitions of various types of Contribution Links. Contribution Links can be used to describe the contribution of Beliefs on Softgoals. Also, Beliefs can receive Contribution Links from other Beliefs. Even though Beliefs can receive Contributions from other elements such as Tasks, this has not been widely used in i* models.



4.2.2.6 Guideline (Beginner,Layout) Avoid overlapping elements inside or outside Actors. Discuss

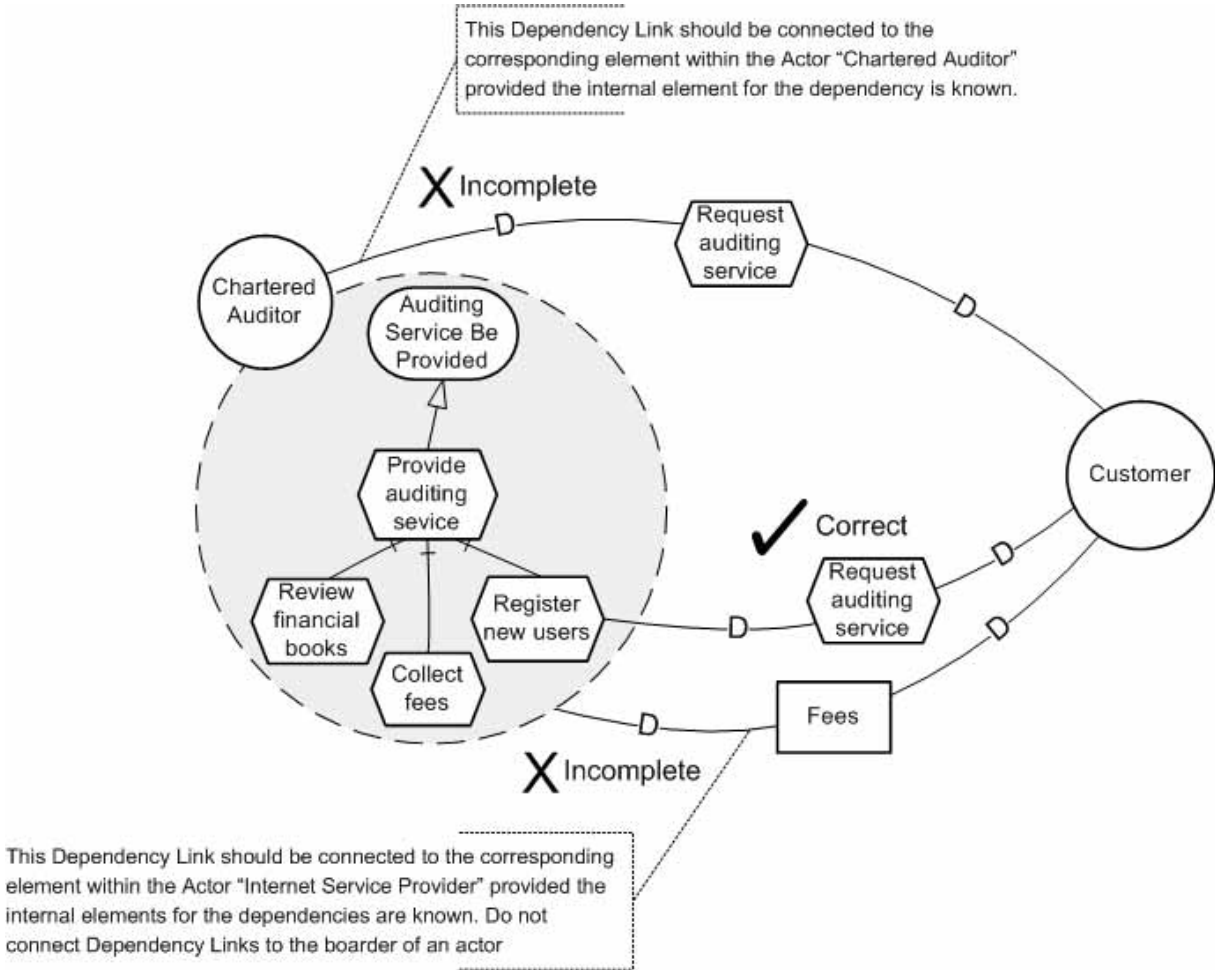
Discussion: Overlapping element within or outside Actors introduces complexity or clutter, which can worsen the readability and understandability of the model. The objective of modeling is better requirements analysis and communication. Overlapping elements make it more difficult for stakeholders to understand and use the model.



4.2.2.7 Guideline (Beginner,Layout) Connect each Strategic Dependency Link in an SR model to the correct element within the actor. Discuss

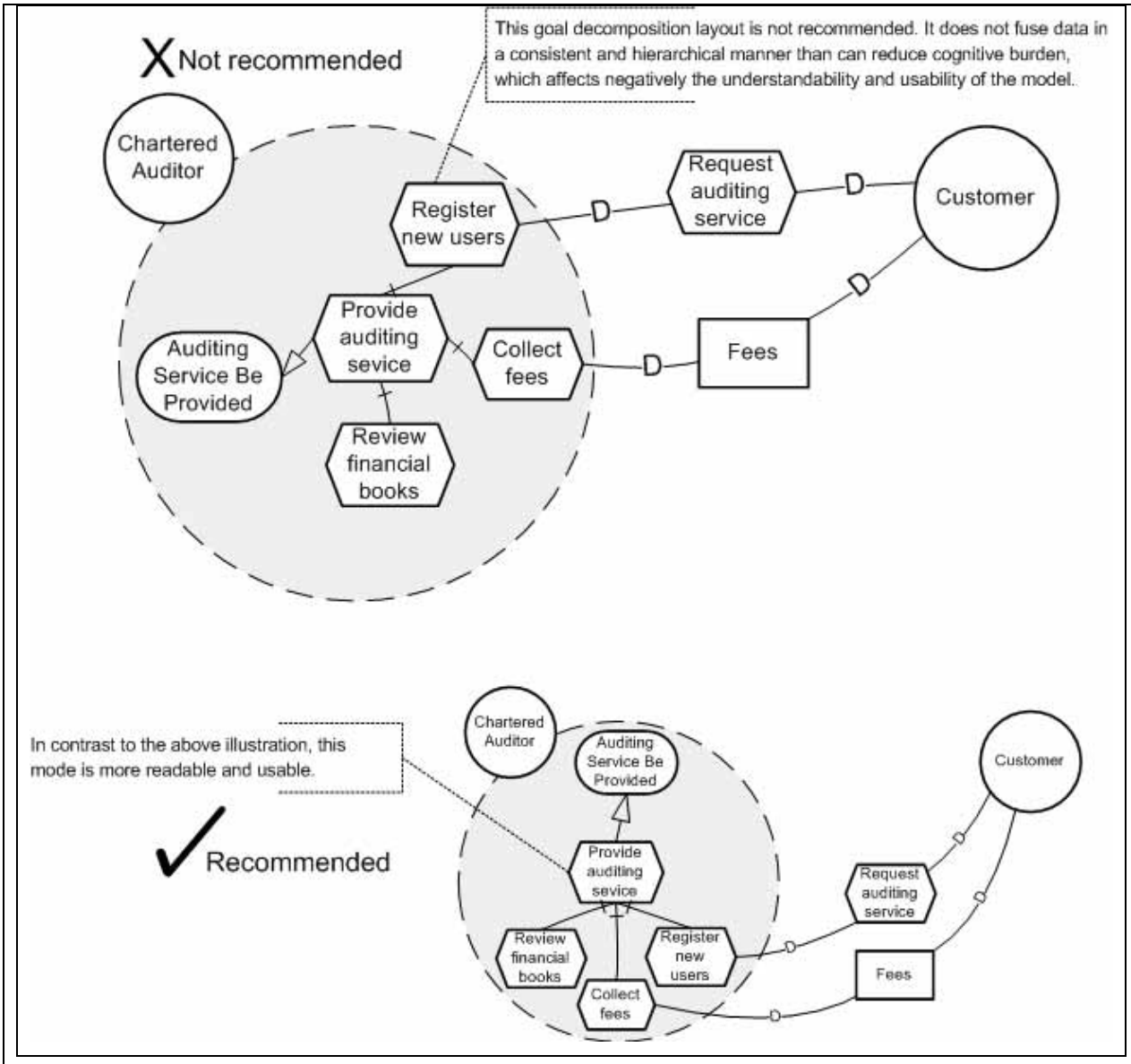
Discussion: The purpose of SR models is to show and analyze how the dependencies are satisfied

between Actors. Connecting the Dependency Links their internal elements of the dependency helps in achieving the objective of SR models. In some cases, however, when the model is incomplete (in progress) and the internal element for a dependency is not known, the Dependency Link may remain connected to the Actor, as in the SD model, or remain connected to the Actor boundary. As shown in the illustration, when the internal elements of an actor are known, the Dependency Links are connected to the correct, internal elements, not to the Actor or Actor boundary. Applying this guideline helps in developing more accurate and usable models.



4.2.2.8 Guideline (Beginner,Layout) **Adopt or follow a consistent direction for the goal refinement/decomposition hierarchy as much as possible.** **Discuss**

Discussion: Adopting a consistent direction for the goal refinement and decomposition hierarchy helps in improving the overall readability, understandability, and therefore the usability of the models. Also, using such hierarchical structure allows modelers to fuse data and reduce cognitive burden on the users of the model by breaking higher level main goals into lower level sub-goals.



4.2.2.9 Guideline (Beginner,Layout) **Do not draw SR model elements outside the boundaries of the corresponding actors. Discuss**

Discussion: If Intentional elements, such as Goals and Softgoals of Actors are meant to by Goals and Softgoals of an Actor, they should be drawn inside the boundaries of the Actors. Intentional elements are extended between Actors only via Dependency Links. Figure 1 depicts a correct i* notation of the internal elements.

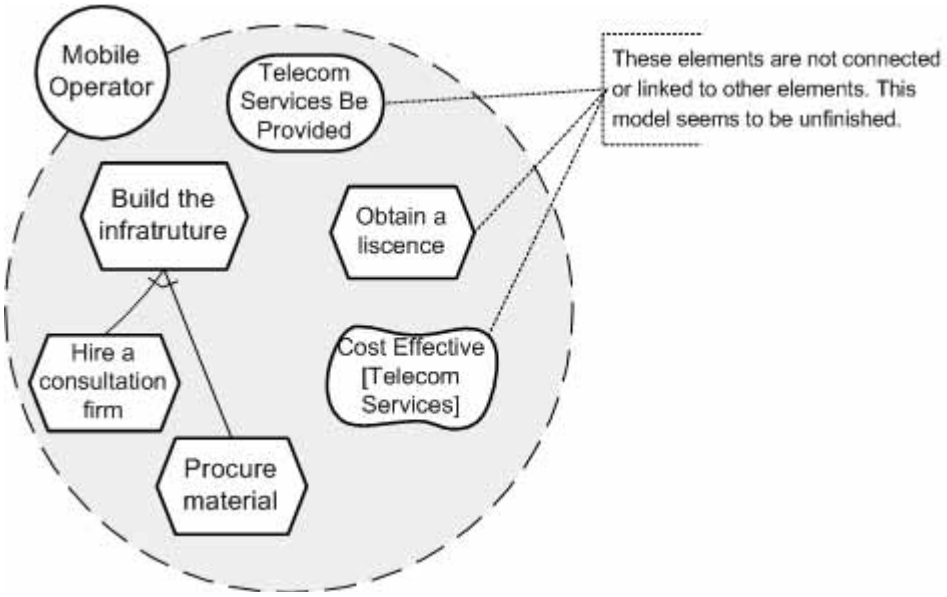
Note: In Tropos, internal elements may be shown partially attached and overlapping with an Actor. Figure 2 illustrates an example. This notation is not used in i*.

4.2.2.10 Guideline (Beginner,Layout) **Unconnected elements within an Actor is indicative of an**

incomplete model. Discuss

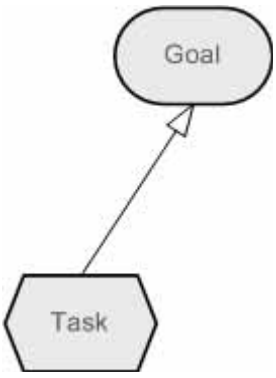
Discussion: Part of the objectives of an i* SR model is to help in analyzing the fulfillment and satisfaction of intentionality elements of and Actor and in expressing the rational as related to the Strategic Dependencies between the Actors. Leaving SR elements without proper connection to related intentional elements within an actor might introduce some difficulty in the readability, understandability, and applicability of the final model.

In some other cases such as when dealing with complex models in research work, the modeler might decide to keep some elements (especially Goals and Softgoals) of an Actor without Links to other elements to indicate the presence of a Goal, for example, that has not yet been operationalized or addressed and to work as reminder so as not to forget it.



4.2.3 Means-Ends Links

These links indicate a relationship between an end, and a means for attaining it. The “means” is expressed in the form of a task, since the notion of task embodies how to do something, with the “end” is expressed as a goal. In the graphical notation, the arrowhead points from the means to the end.



4.2.3.1 Guideline (Beginner,Concept) Means-Ends are only used to link a Task to a Goal. Discuss

Discussion: The only place where a Means-Ends link can be used is between a Task and a Goal. This type of link cannot be used between Tasks, Softgoals, or between any other combinations of links. Refer to Guideline 4.2.2.1.7 for additional discussion on using Means-Ends with Goals. Figures 1 and 2 illustrate some possible scenarios of incorrect Means-Ends connections.

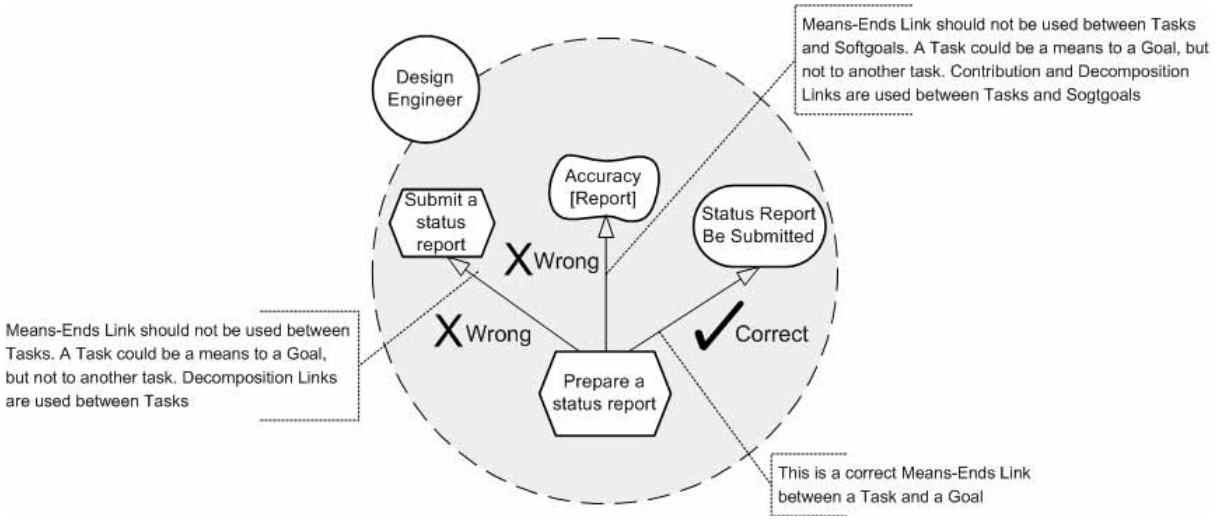


Figure 1 Examples of correct and some wrong Means-Ends links

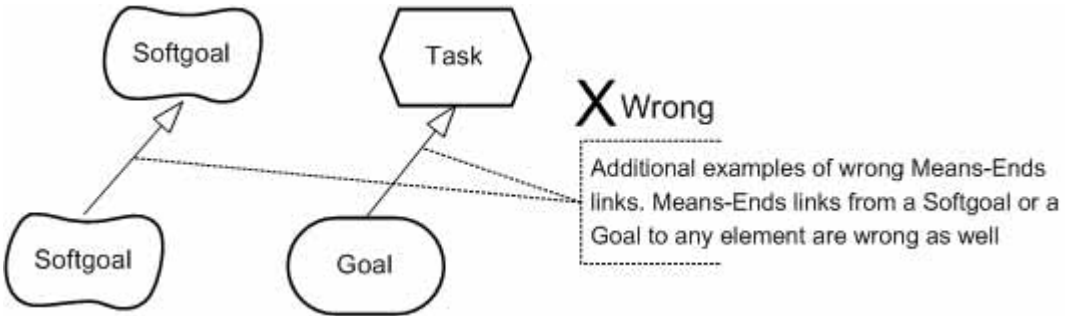


Figure 2 Additional examples of wring Means-Ends links

4.2.4 Decomposition Links

A task element is linked to its component nodes by decomposition links. A task can be decomposed into four types of elements: a subgoal, a subtask, a resource, and/or a softgoal - corresponding to the four types of elements. The task can be decomposed into one to many of these elements. These elements can also be part of dependency links in Strategic Dependency model(s) when the reasoning goes beyond an actor's boundary.

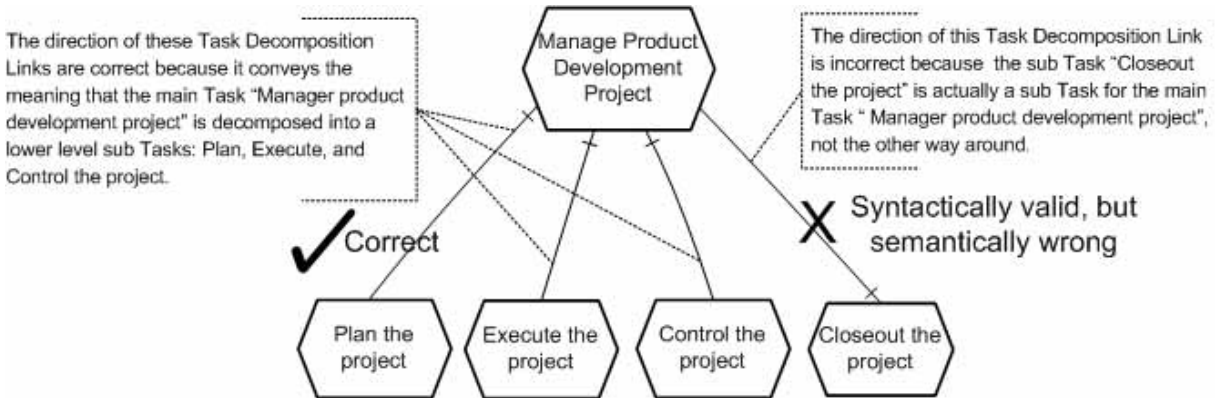
- Task-Goal Decomposition: Subgoal. In this kind of decomposition it is not specified how the goal is to be achieved, allowing alternatives to be considered.
- Task-Task Decomposition: subtask. When a task is specified as a subcomponent of a (higher) task, this restricts the higher task to that particular course of action.
- Task-Resource Decomposition: resourceFor: The entity represented by the resource is not

considered problematic by the actor. The main concern is whether it is available (and from whom, if it is an external dependency).

- Task SoftGoal? Decomposition: softgoalFor: When a softgoal is a component in a task decomposition, it serves as a quality goal for that task, and thus guides (or restricts) the selection among alternatives in further decomposition of that and other tasks.

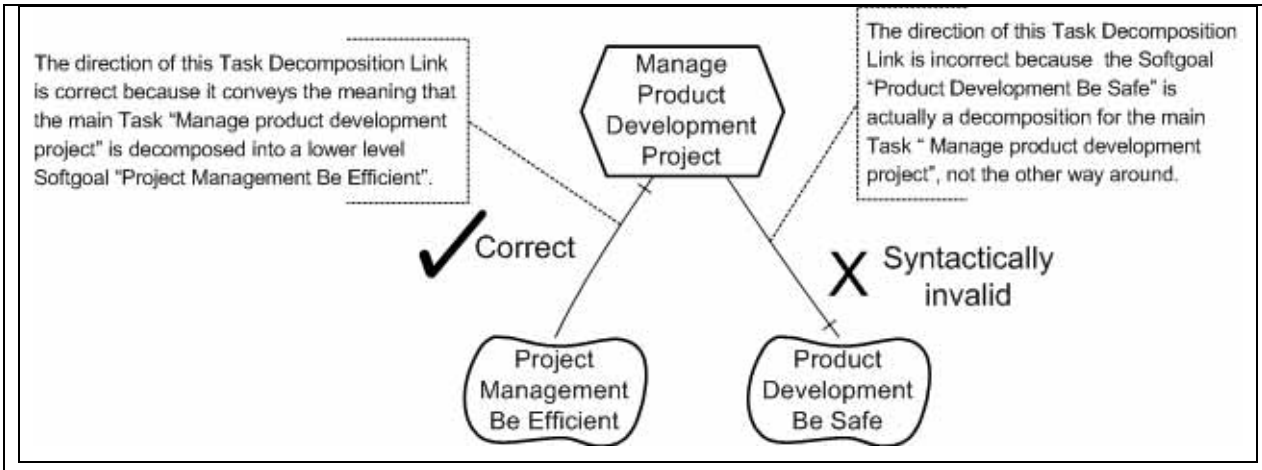
4.2.4.1 Guideline (Beginner,Concept) Be consistent with the direction of the Task Decomposition Link between a Task and sub Task or Resource. Discuss

Discussion: The direction of the Task Decomposition Link should be from the main Task to the sub Task. This type of link is used within Actors to decompose a task into sub Tasks as shown in the illustration or into other elements such as Resources, Goals, and Softgoals. Syntactically, there is nothing wrong with illustration model, but it is semantically wrong and illustrates a poor layout.



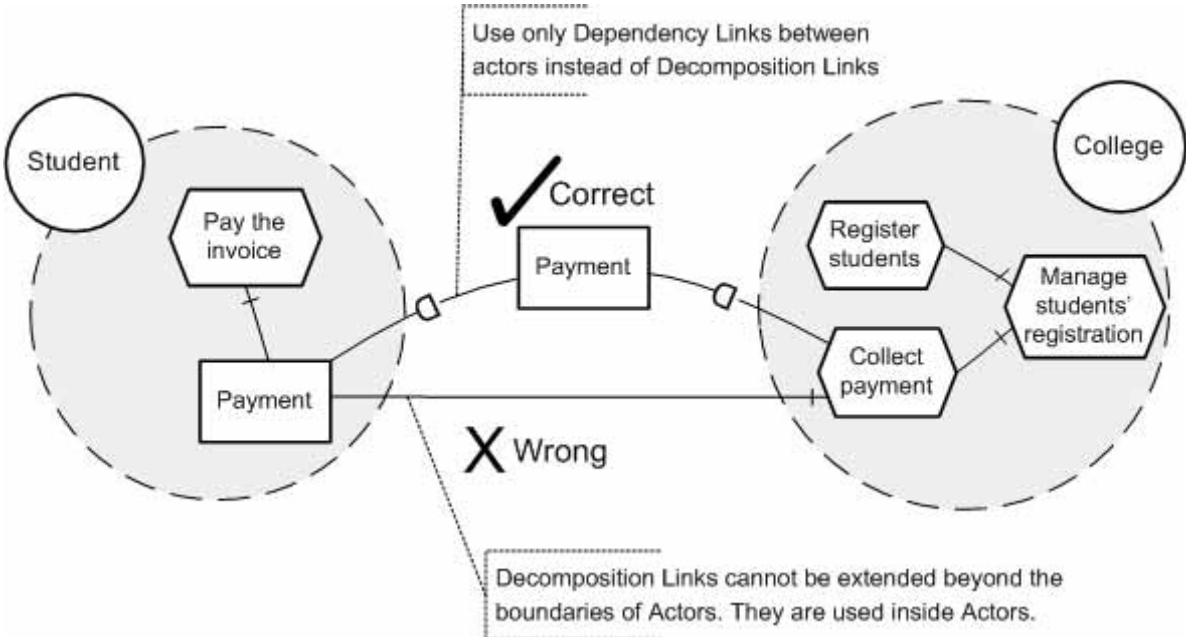
4.2.4.2 Guideline (Beginner,Concept) Be consistent with the direction of the Task Decomposition Link between a Task and a Softgoal. Discuss

Discussion: The direction of the Task Decomposition Link should be from the main Task to the Softgoal. This type of link is used within Actors to decompose a task into Softgoals as shown in the illustration or into other elements such as Tasks, Resources, and Goals. In this illustration's particular example, the Softgoal component "Project Management Be Efficient" is a quality attribute or non-functional requirement that has been introduced for the Task "Manage Product Development Project". This Softgoal component can be refined further into other Tasks or Softgoals, which can me be used to guide the selection among alternatives for the main Task and the sub Tasks.



4.2.4.3 Guideline (Beginner,Concept) **Do not extend Decomposition Links beyond the boundaries of actors. Discuss**

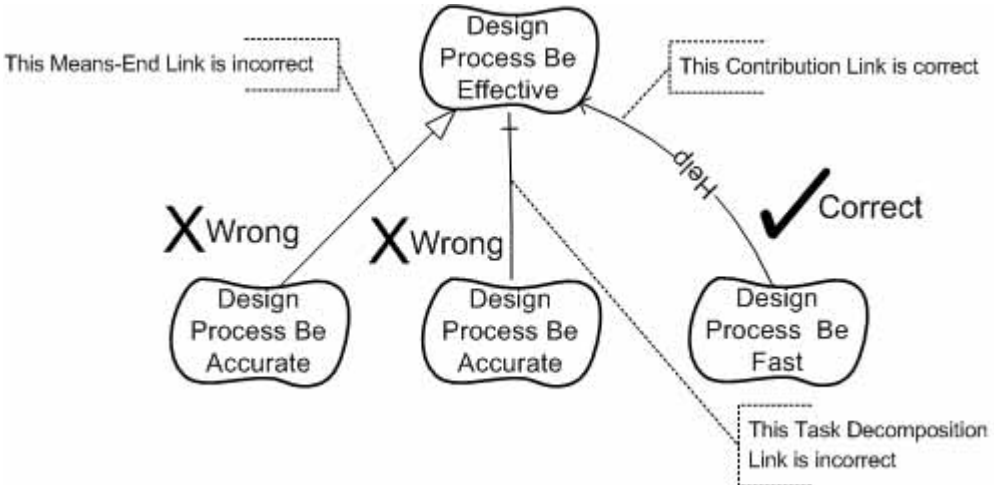
Discussion: Decomposition Links need to be drawn within an Actor, and not between Actors. Dependency Links should be used across Actors. These Dependency Links prompt the modeler to ask: What does Actor "Student" depend on Actor "College" for? And what is the nature of the dependency? The Answer is reflected in the Dependums, as these Dependums inform about the type of Dependency Links that are required by the modeler. Section 4.1.3 "Strategic Dependencies" discusses various types of Dependency Links that can be used between Actors.



4.2.4.4 Guideline (Beginner,Concept) **Don't use the Task Decomposition Link or Means-End Link to refine Softgoals. Discuss**

Discussion: Softgoal refinement requires the use of any of the Contribution Links such as Make,

Some+, Help, Break, Some-, Hurt, Unknown, And, and Or. Means-Ends and Decomposition Links can't be used in this context.

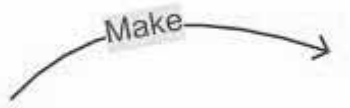


4.2.5 Contribution Links

Elements, which are discussed in Section 4.2.2 "Elements/Nodes" are: Goal, Softgoal, Task, Resource, and Belief. Contribution Links, which are also covered in this section, are: Make, Some+, Help, Break, Some-, Hurt, Unknown, And, and OR. Any of these Contribution Links can be used to link any of the elements to a Softgoal to model the way any of these Elements contributes to the satisfaction or fulfillment of the Softgoal. Goal-oriented Requirement Language (GRL) uses a variation of the i* Contribution Links notation. Refer to Guideline 4.2.5.10 for more information.

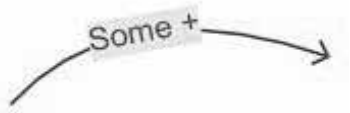
4.2.5.1 Make

A positive contribution strong enough to satisfice a softgoal.



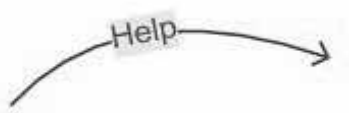
4.2.5.2 Some+

Either a make or a help contribution, a positive contribution whose strength is unknown.



4.2.5.3 Help

A partial positive contribution, not sufficient by itself to satisfice the softgoal.

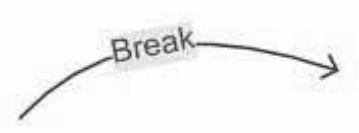


4.2.5.4 Unknown

A contribution to a softgoal whose polarity is unknown.

4.2.5.5 Break

A negative contribution sufficient enough to deny a softgoal.

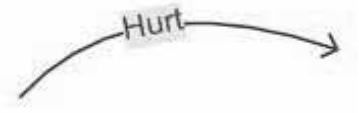


4.2.5.6 Some-

Either a break or a hurt contribution, a negative contribution whose strength is unknown.

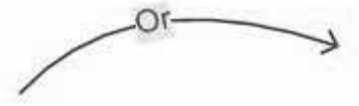
4.2.5.7 Hurt

A partial negative contribution, not sufficient by itself to deny the softgoal.



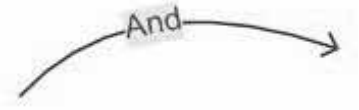
4.2.5.8 Or

The parent is satisficed if any of the offspring are satisficed.



4.2.5.9 And

The parent is satisficed if all of the offspring are satisficed.



4.2.5.10 Guideline (Beginner,Concept) Use Contribution Links from any element only to a Softgoal element. Discuss

Discussion: Contribution Links are not allowed from any element to a goal, only to Softgoals. Please refer to Section 1.4.2.5 “Contribution Links” for an explanation about Contribution Links. Figure 1 depicts only two possible cases of wrong uses of Contribution Links (from a Task to a Goal and from a Task to a Task). There are other wrong cases, however, that the Contribution Link should not be used in such as Goal to Goal, Goal to Task, Softgoal to Goal, and Softgoal to Task.

Figure 2 illustrates all the possible correct cases of using Contribution Links to a Softgoal element using i* notation.

Figure 3 depicts the GRL (Goal-oriented Requirement Language) variation of the i* Contribution Links notation.

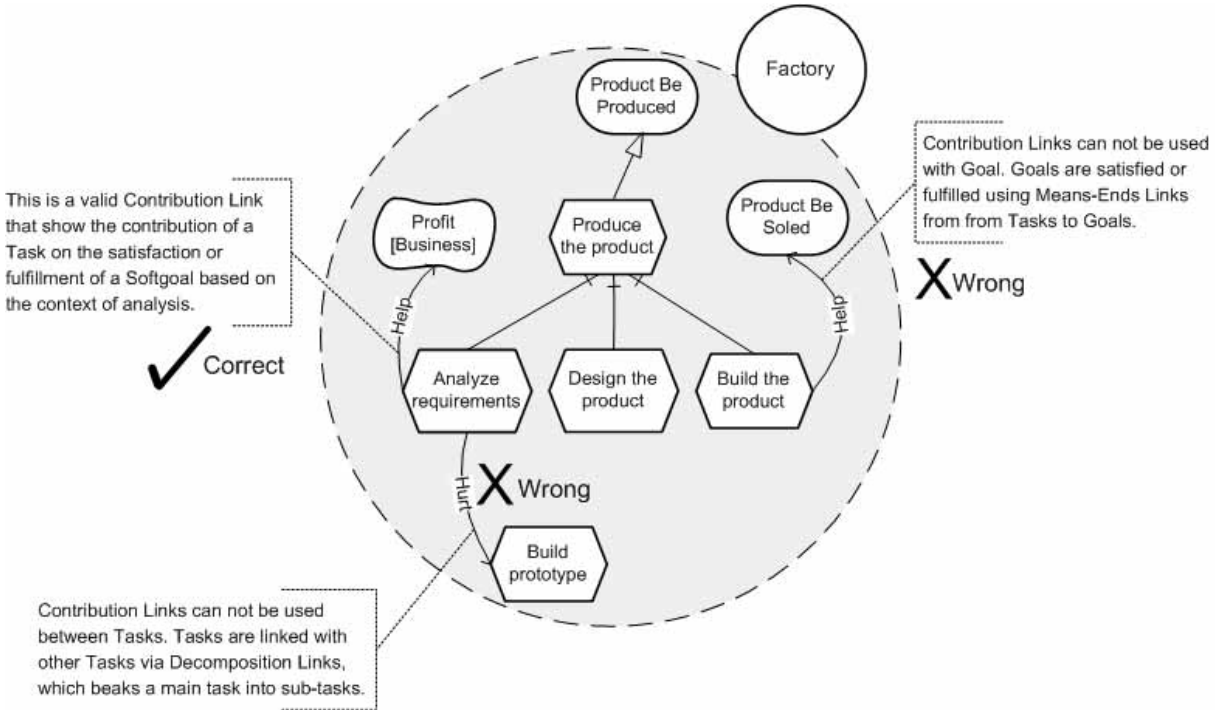


Figure 1 An example of some possible correct and wrong uses of Contribution Links

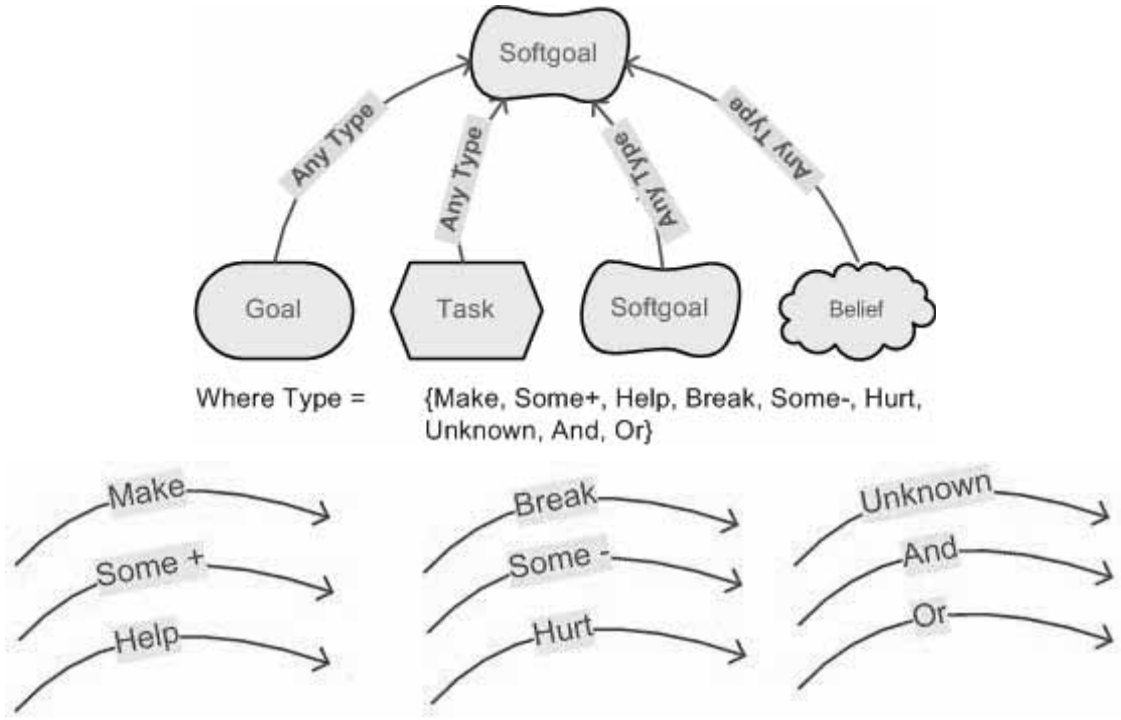
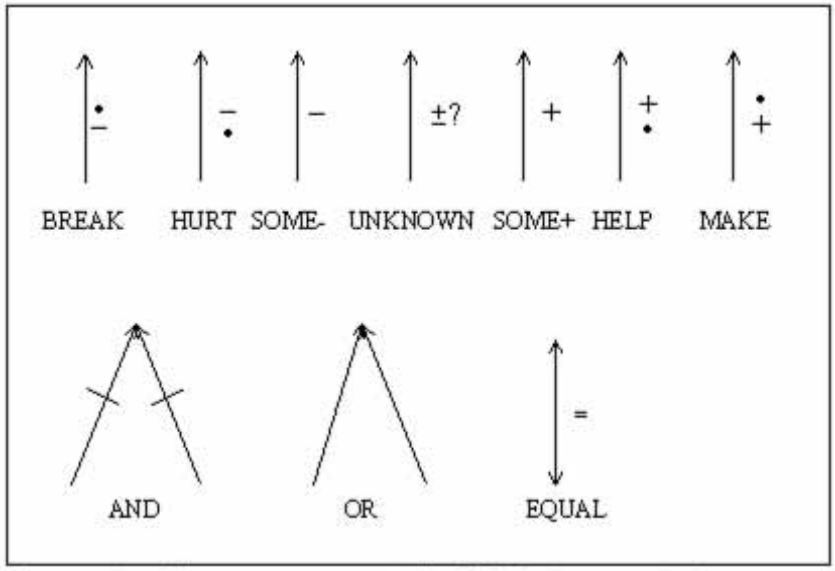


Figure 2 Contribution Links to a Softgoal element using i* notation

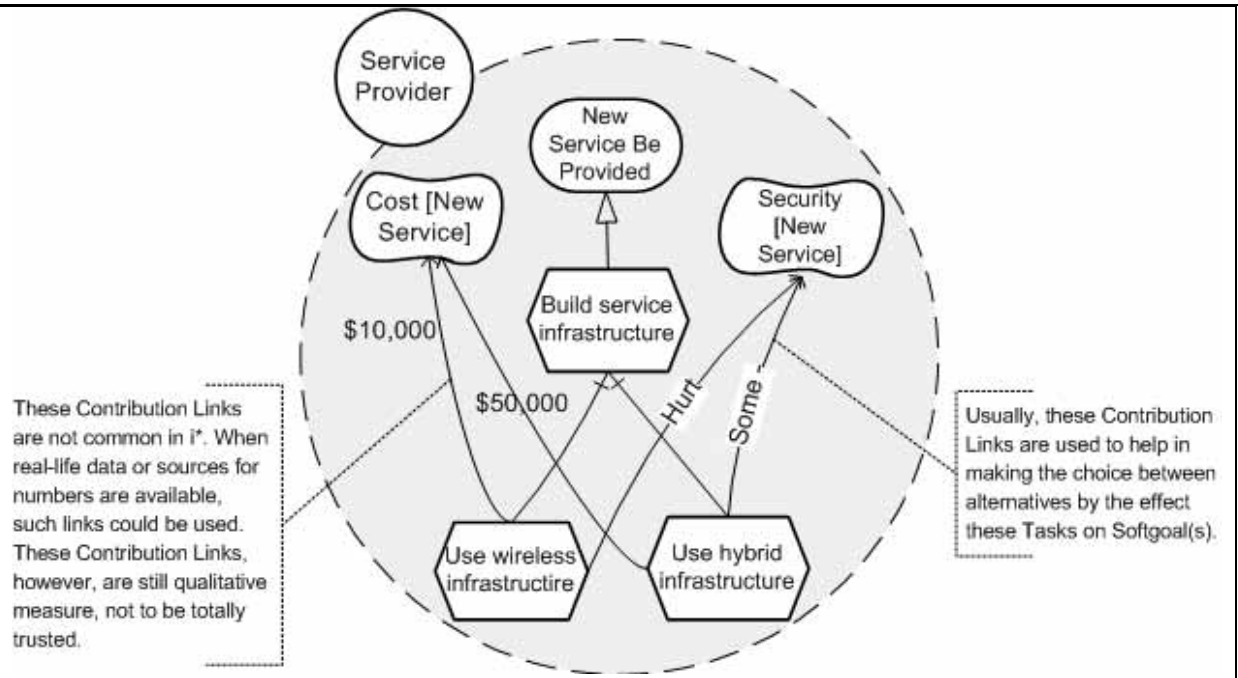


Source: http://www.cs.toronto.edu/km/GRL/grl_syntax.html

Figure 3 GRL variation of the i* Contribution Links notation

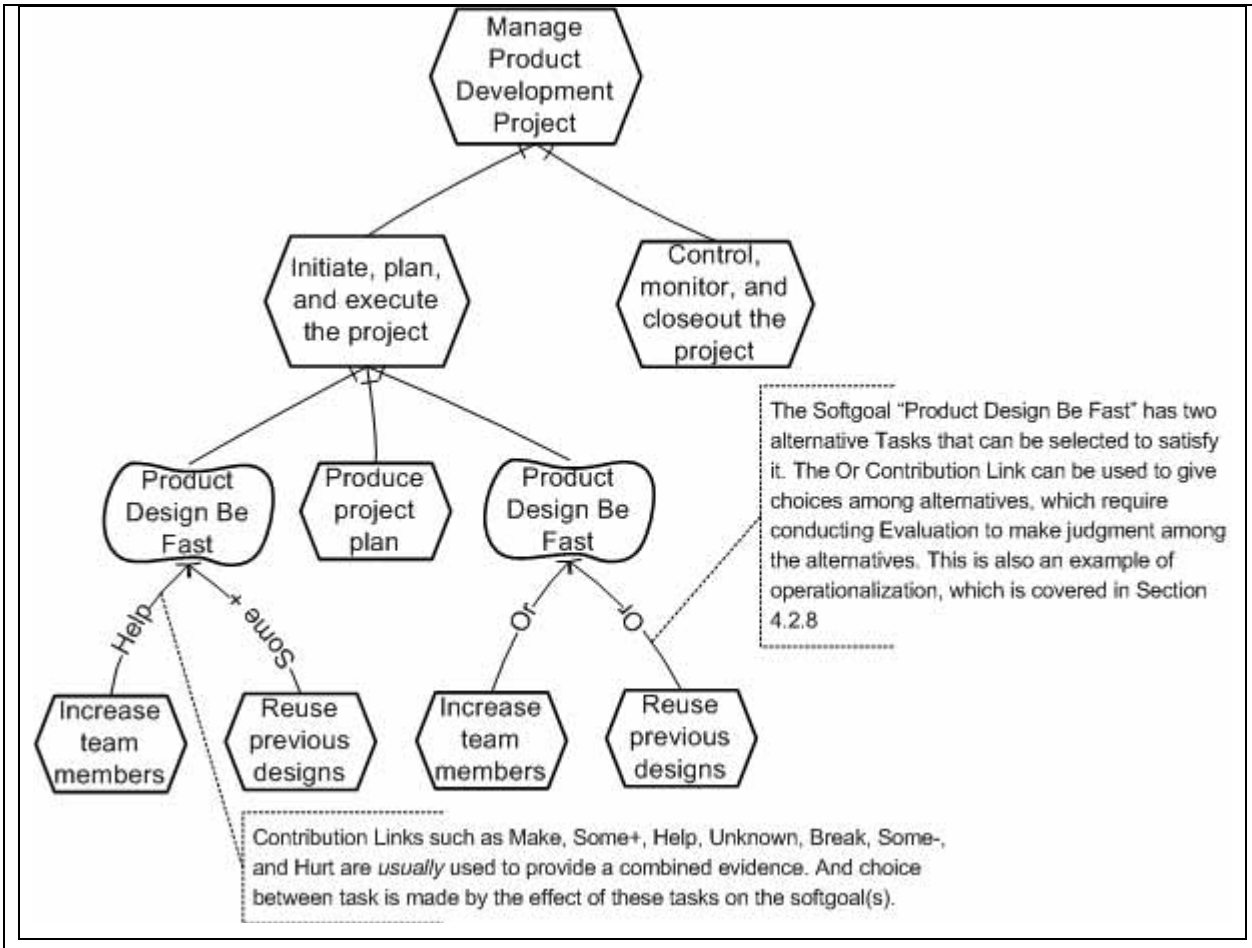
4.2.5.11 Guideline (Beginner,Concept) Avoid introducing ad hoc or improvised link types. If you must, define their syntax and semantics as extensions to i*. Discuss

Discussion: Usually, Contribution Links are used to evaluate the satisfaction of Softgoals. In some instances, however, quantitative Contribution links labels (such as numbers or different symbolic types) can be used if real-life data or sources of numbers are available to evaluate contributions of alternative Tasks on the satisfaction of Softgoals. Otherwise, adding numbers is just making a finer grained qualitative evaluation measure. The user of the model or the modeler has to be aware that it is still a qualitative measure, not to be totally trusted. Additionally, should the modeler need to introduce new link types, their syntax and semantics must be defined.



4.2.5.12 Guideline (Beginner,Concept) Use the OR Contribution Links to indicate alternatives for satisficing a Softgoal. Discuss

Discussion: The Or Contribution Link can be used, even though very rarely, to select among alternative tasks to satisfice a Softgoal. This Link means that an Actor has the choice of making a selection for satisficing the quality attribute or the non-functional requirement (Softgoal). This process of selecting among alternatives is followed by an evaluation process to make judgment on the impact of each alternative on other Softgoals in the model. Alternatively, Contribution Links such as Make, Some+, Help, Unknown, Break, Some-, and Hurt are usually used to provide a combined evidence. A choice between tasks is made by the effect of these tasks on the Softgoals itself and other Softgoals.



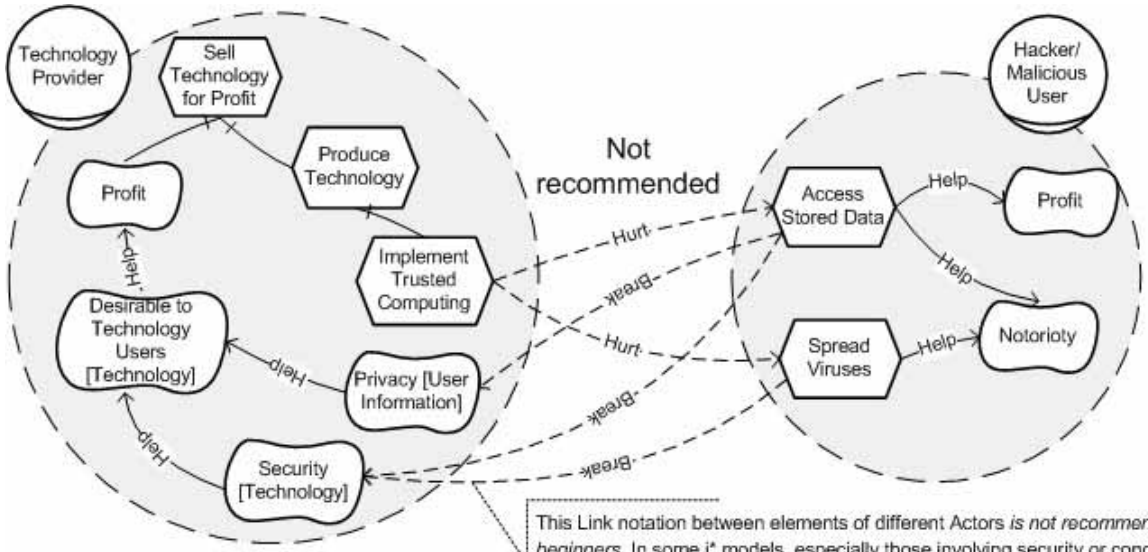
4.2.5.13 Guideline (Beginner,Concept) Don't use Correlation or Contribution Links between actors. Discuss

Discussion: Dependency relationships between Actors should be represented in SD and SR models using the standard Dependency Links notation. These links should reflect all types of dependency relationships between Actors. Contribution Links (solid lines) and Correlation Links (dotted lines) should not be used between Actors. In some i* models, especially those involving security or concepts discussed in advanced research, there is a need to show attacks and defense scenarios using correlations and contributions between Actors. They add new expressive power. These correlations and contributions between Actors are treated as extensions to i*.

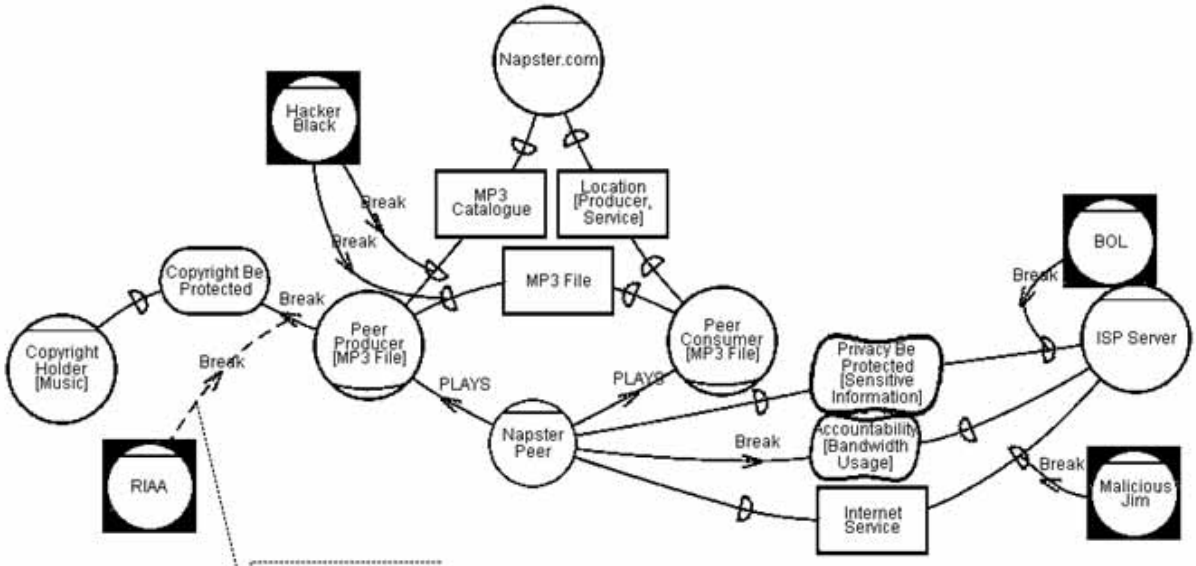
Examples 1 and 2 depict some exceptional scenarios where Correlation and Contribution Links (solid or dotted lines) have been used between Actors to provide more expressiveness power. It is recommended that the beginners and students do not break the general rule or the guideline of not using Correlation and Contribution Links between actors.

Additionally, Correlation Links have been used in the NFR Framework's examples. These links basically have the same syntax as Contribution Links, from any element only to Softgoals, but drawn in dotted lines. Correlation Links represent side effects from an element (such as a Task or a Softgoal)

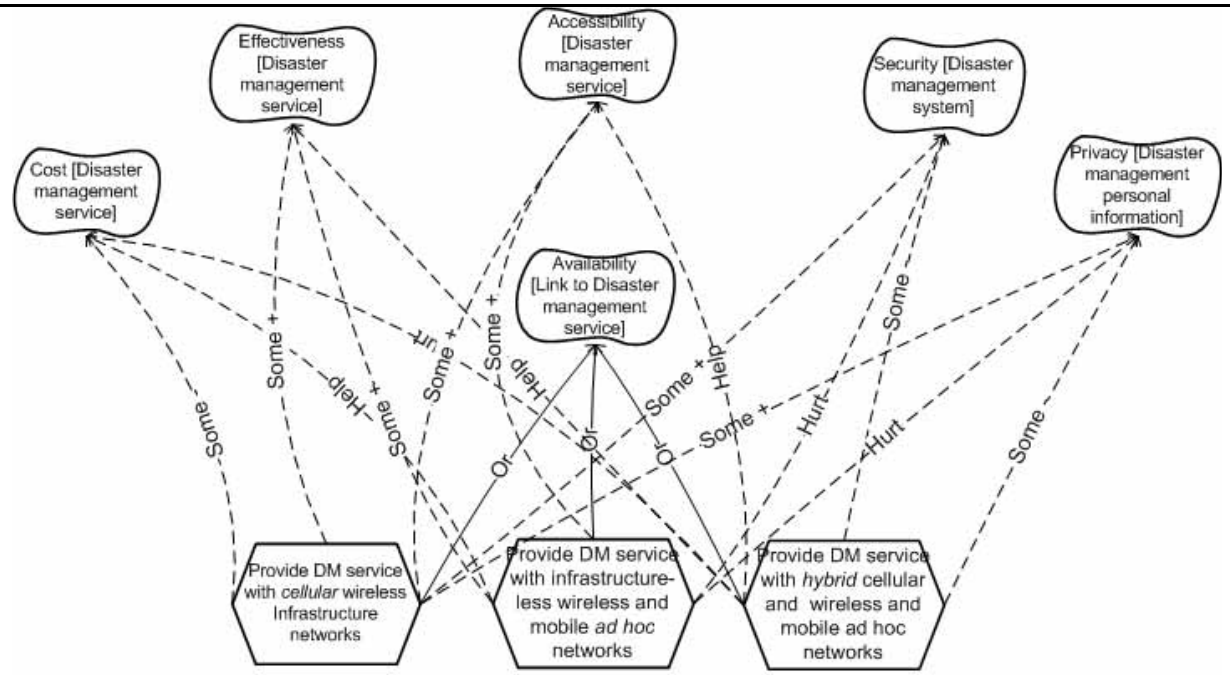
to a Softgoal. Example 3 depicts an NFR Framework using i* notation to illustrate the concept of codification of design knowledge for reuse and building NFR catalogues. The example shows the Operationalization of a Softgoal and its correlations (side effects) on other domain related Softgoals.



Example 1 Reproduced from PST06_Trust Technology (Horkoff, Yu, and Liu, 2006)



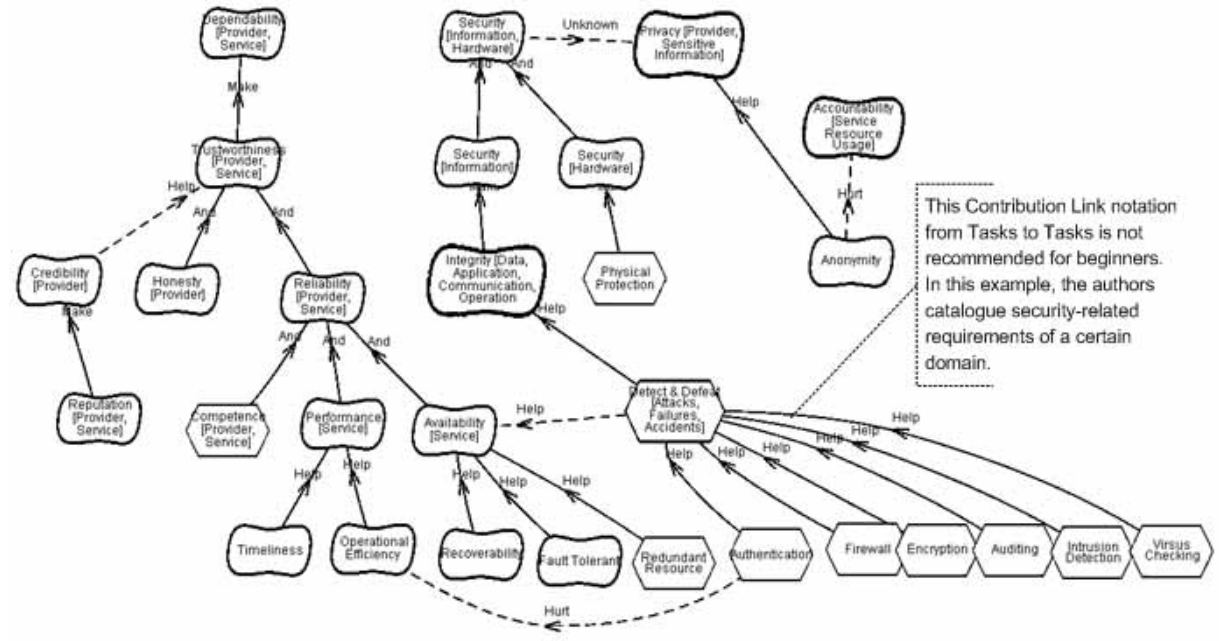
Example 2 Source: SREIS02-Sec (Liu, Yu, and Mylopoulos, 2002)



Example 3 An example of i* correlation notation using NFR Framework

4.2.5.14 Guideline (Beginner,Concept) Don't use Correlation or Contribution Links from a Task to a Task. Discuss

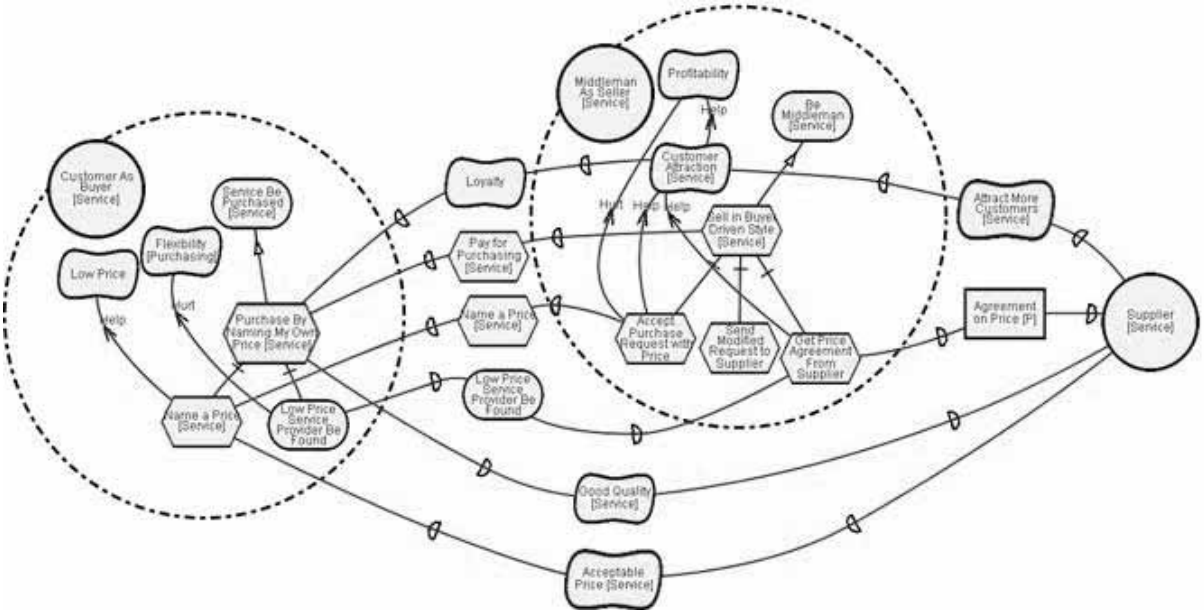
Discussion: Correlation or Contribution Links are used only from any element to a Softgoal. In some instances advanced modelers might bypass this rule in Security analysis or other advanced research topics that require adding new expressive power. Therefore, it is recommended for beginners and student to follow the guideline.



Source: SREIS02-Sec (Liu, Yu, and Mylopoulos, 2002)

4.2.6 Leaf Elements

Elements in i* SR models which are not decomposed into further elements, or Which do not receive any contribution from any link, are called leaf elements. Leaf elements can be, but are not necessarily operationalizations, as they may not represent high-level executable tasks.



4.2.7 Strategic Rationale Example Model: Buyer Drive E-Commerce from Yu01

4.2.8 Operationalizations and Refinement of Goals and Softgoals

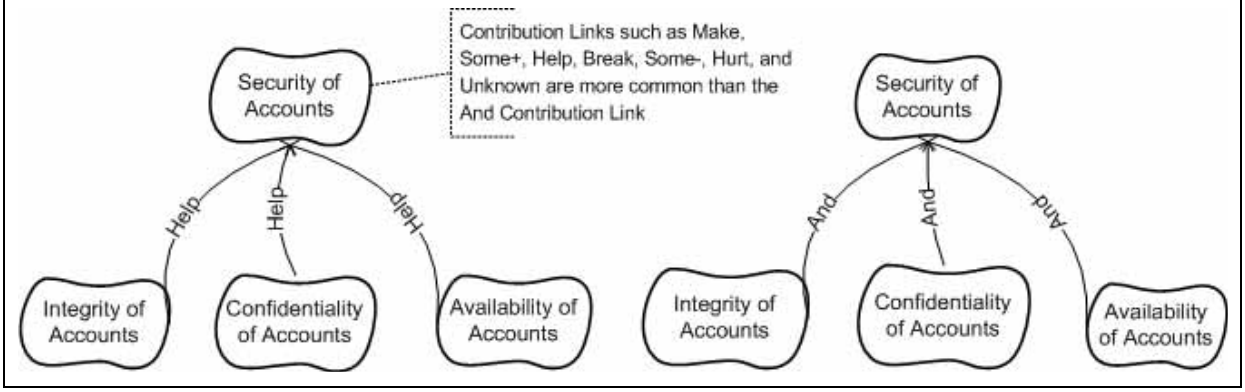
When softgoals are decomposed into tasks, such tasks are called operationalizations. They are achievable through a concrete action. Operationalizations can be further decomposed into further, more specific operationalizations or even to other i* elements such as goals and softgoals, necessary to achieve the tasks.

Goals and Softgoals are intentional elements that could be used externally in the Dependency Links between Actors and internally within Actors. Depending on some reasons such as the required level of analysis, required efforts, complexity, and objectives of the models, internal high level Goals and Softgoals could be refined (broken down) into lower level goals and Softgoals. These refinement processes in effect, define and elaborate on the meaning of the goals and make them more concrete. Some guidelines are provided in this section to help in making the refinement process more systematic and usable.

4.2.8.1 Guideline (Beginner,Concept) Use Contribution Links to decompose or refine a broad softgoal or non-functional requirement (NFR) into smaller components. Discuss

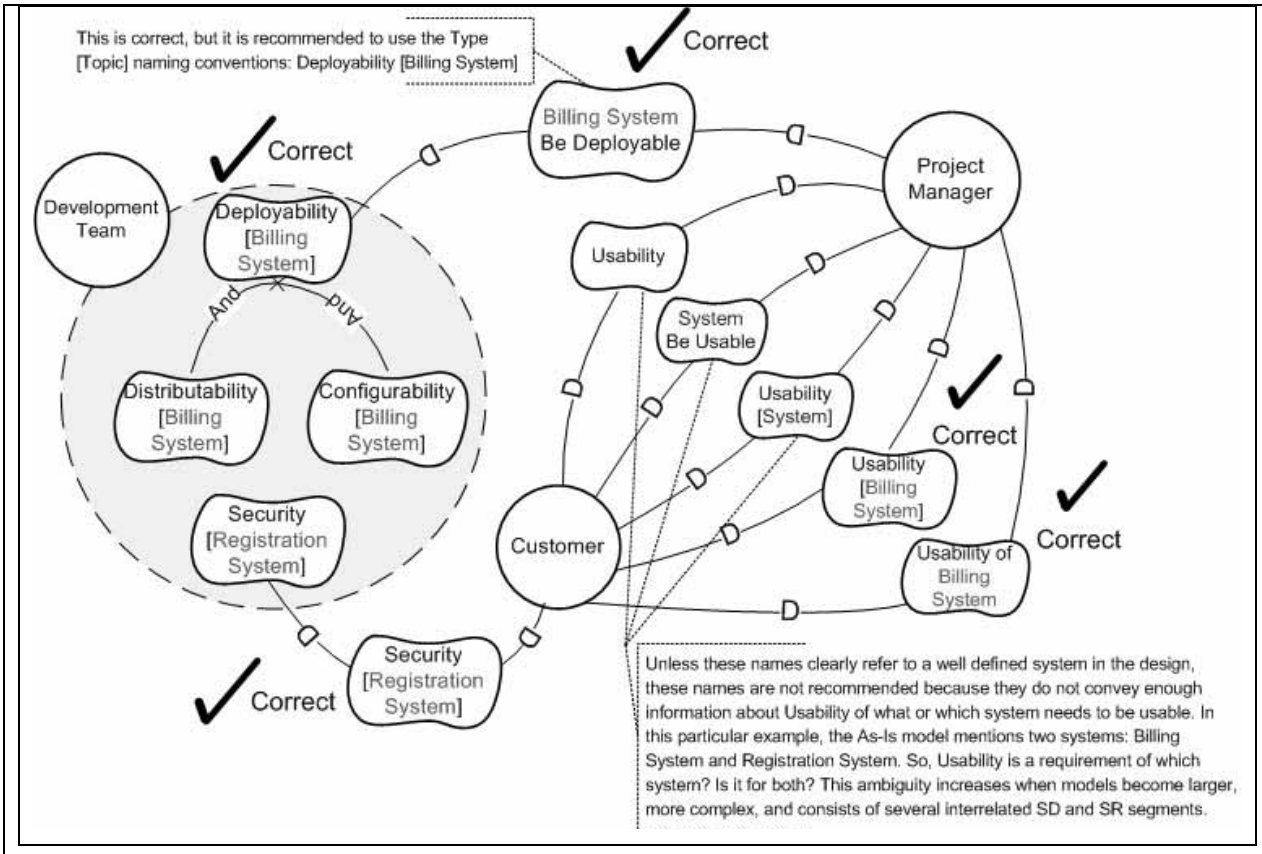
Discussion: The decomposition and refinement of a very broad or abstract softgoal helps in making it more concrete. For example, the Security NFR is a broad quality, which can be refined, for example,

into: Integrity, Confidentiality, and Availability. The And Contribution Link in this example means that all of the three components are part of the definition of the Security NFR. Advanced modelers, however, might find using the And Contribution Link restrictive because of the problem that all Softgoals must be satisficed to have any positive value for the Security NFR (in this example) and usually a modeler might want to show a partial value if only one or two of the sub goals are satisfied. Therefore, advanced modelers might use multiple Help Contribution Links instead of the And Contribution Links. Furthermore, the type and/or topic of a Softgoal or NFR can be decomposed or refined. The guidelines and discussions for refining the type and topic of a Softgoal are discussed in the subsequent guidelines in this section.



4.2.8.2 Guideline (Beginner,Naming) **To facilitate systematic refinement, use Type and Topic naming convention for Softgoals.** **Discuss**

Discussion: It is recommended to use the Type [Topic] naming convention for Softgoals to facilitate systematic Softgoal refinement, eliminate model misunderstanding and ambiguity, maintain naming consistency within the model and among multiple models, and provide a means of promoting knowledge codification and cataloguing. In the initial stage of modeling an As-Is system, the modeler, however, may want to retain the terminology and reasoning structure of stakeholders from, for example, interview raw data. In this case the Type [Topic] naming convention may not be appropriate.



4.2.8.3 Guideline (Beginner,Naming) **Where Softgoals are named according to the Type and Topic structure, be consistent in each Softgoal refinement step to refine either by Type or by Topic.** **Discuss**

Discussion: High level Softgoals should be refined into lower level Softgoals, and eventually operationalized. The Type or Topic of a Softgoal or NFR can be refined into lower level Softgoals or NFRs. Additionally, consistent refinement of the Type or Topic of a Softgoal helps in producing consistent and systematic because it prompts modelers to consider all applicable Softgoal types for a given topic, or for a given Softgoal type, what sub-topics it is applicable to. Type [Topic] refinement process is to be done on Softgoal's type or topic, but not both at the same time. Please refer to Chung, Nixon, Yu, and Mylopoulos (2000) for a detailed discussion about the NFR framework.

Figure 1 shows an example of refining a Softgoal using Type and Topic structure.

Figures 2 and 3 depict counter examples where the Type and Topic refinement is not consistent. For example, Figure 2 illustrates mixing or introducing a new Softgoal Type "Availability" within the refinement of the original Softgoal, "Transparency". Figure 2 illustrates mixing or introducing a new Softgoal Topic "[Health Record]" within the Topics of the original Softgoal's Topic "[Accounts]".

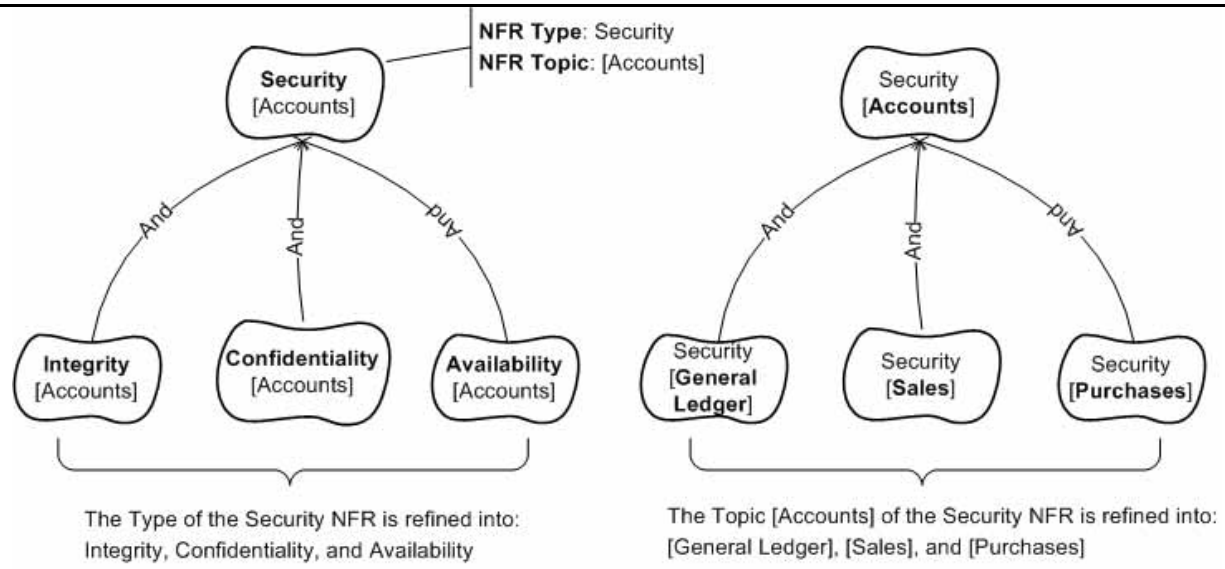


Figure 1 An example of Type and Topic refinement

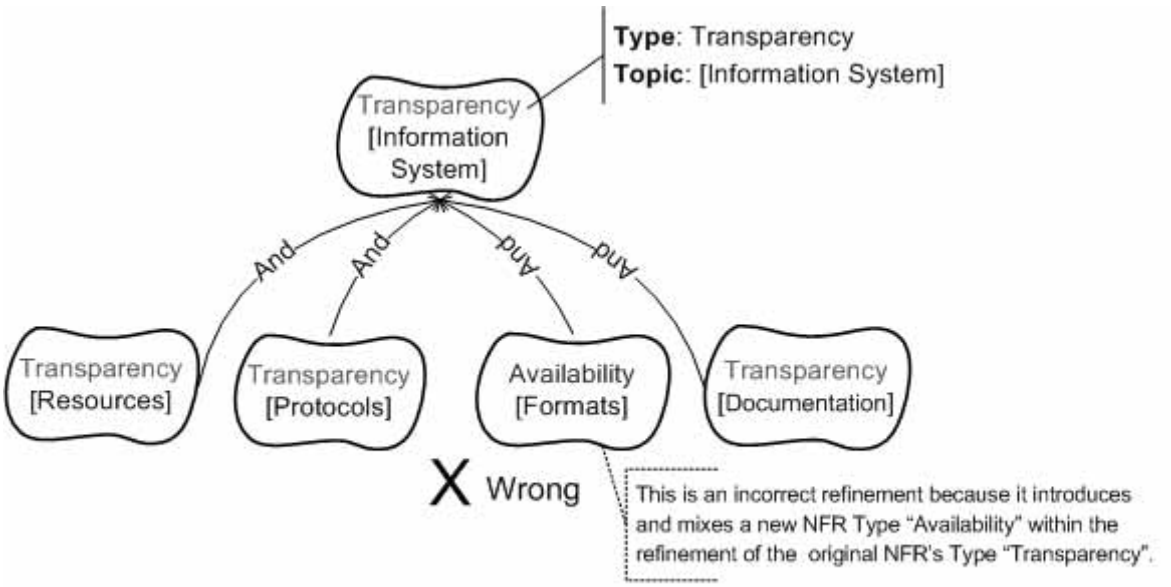


Figure 2 An example of mixing Types within Topic in the Softgoal refinement

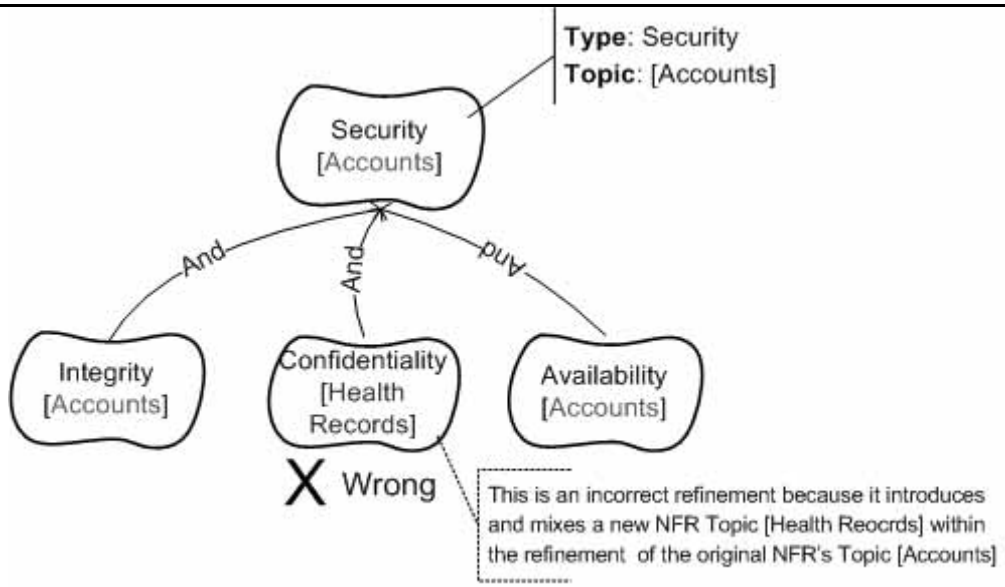


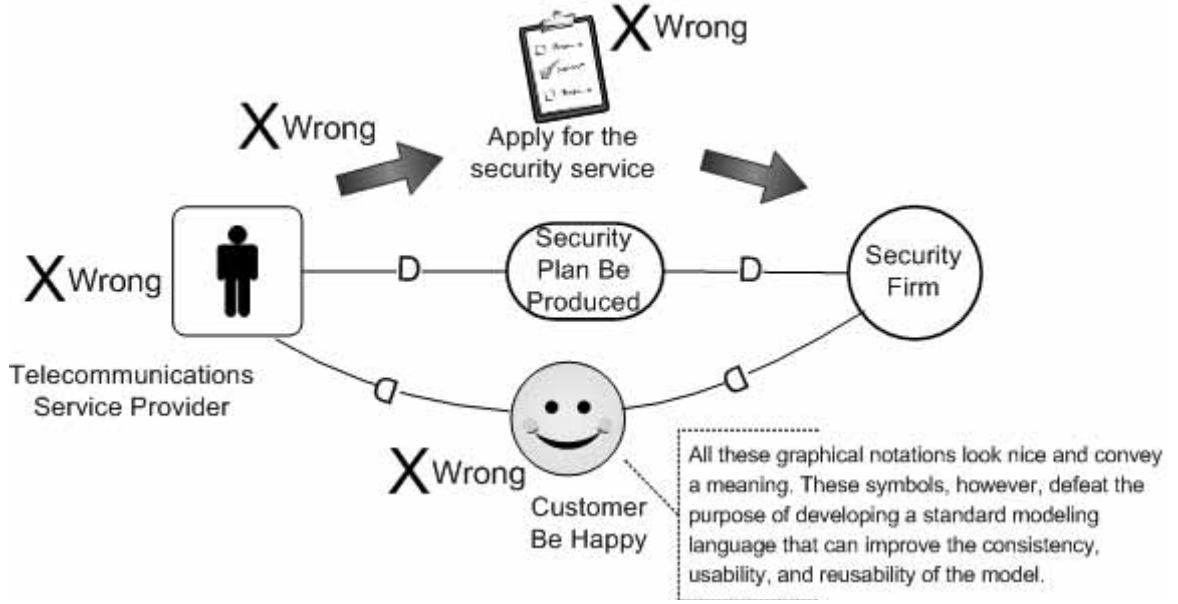
Figure 3 An example of mixing Topics within Types in the Softgoal refinement

4.3 Naming, Icons, and Colors

Labels and names are text strings that are associated with the elements of the model. They are an integral part of the standard i* notation. Non-conventional icons include such things as face and stick figures, artistic icons, and arrows. Coloring could include filling standard elements with certain colors, using specific colors to label parts or the entire model's elements, or use colors to highlight the boundaries of Actors and their elements. Like naming and labeling, using colors and non-conventional icons sometimes can help modelers to convey certain information to the users of the model, but some of the questions that might arise are: what icons and colors should be standardized? In what context they can be used? And how to guarantee that using these non-conventional icons and additional colors are really useable by different types of stakeholders? Therefore, some general guidelines are provided in this section to help in bringing the benefit out of using names, labels, icons, and colors without jeopardizing the readability and usability of the models.

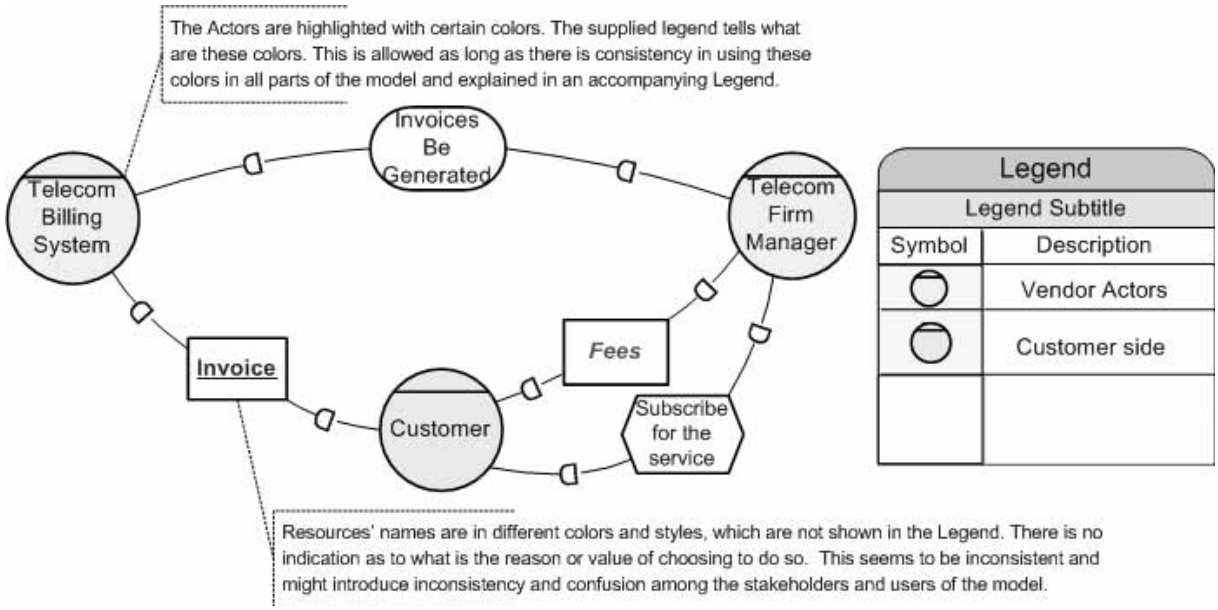
4.3.1 Guideline (Beginner,Naming) Avoid including non standard elements or notations in the model. Discuss

Discussion: Non standard or conventional symbols sometimes defeat the purpose of developing a standard modeling language that can improve the consistency, usability, and reusability of the model. Different analysts may opt to use different symbols and notations to convey the same meaning. This situation creates inconsistency in reading and interpreting the model. This does not imply, however, that expanding i* notation using non-standard syntax is not allowed. New notations might add expressive power. The requirements are that the new notation needs to be clearly explained and the existing i* notation cannot be used instead.



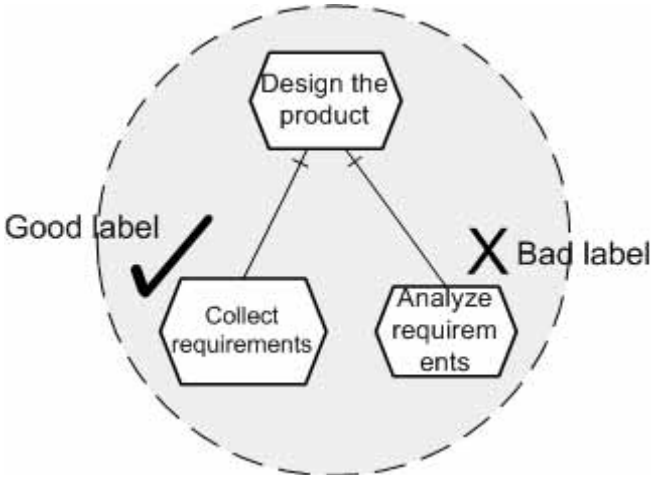
4.3.2 Guideline (Beginner,Naming) Be consistent when using colors in the models. Discuss

Discussion: Using color in models can be useful if used properly and consistently. An accompanying Legend needs to be supplied with the model, if the analyst chooses to do so. The Legend with eliminate any confusion that might arise due to the use of these extra colors in the model. Improper use of colors in the model, however, can lead to issue of consistency and usability. Additionally, a lot of colors can be distracting.



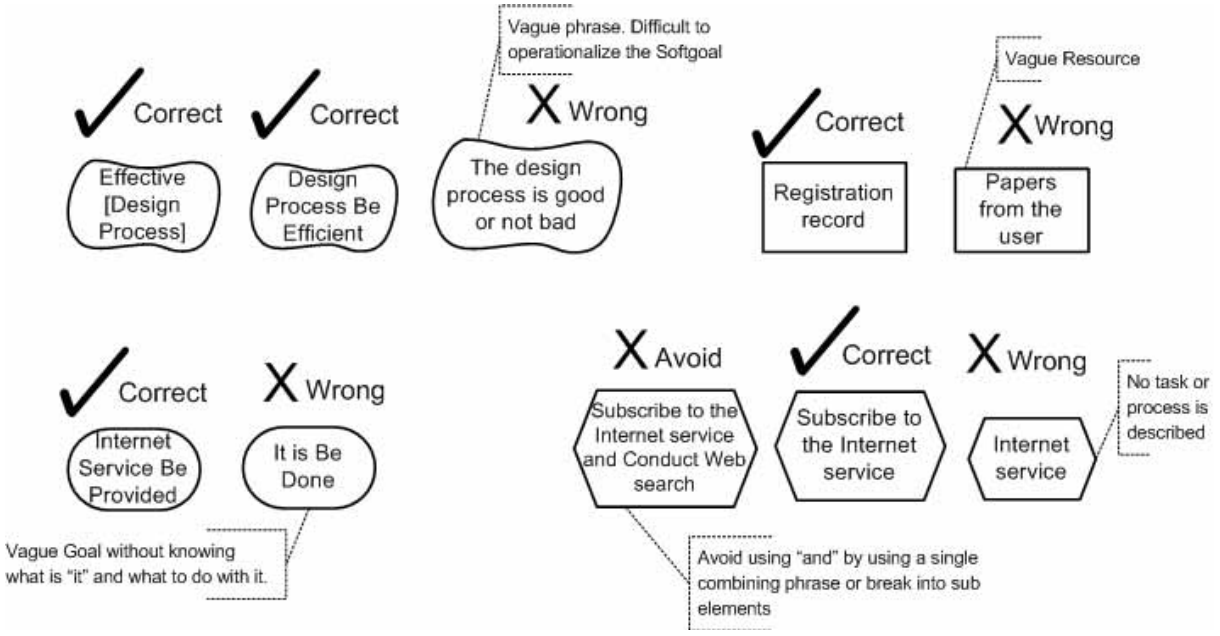
4.3.3 Guideline (Beginner,Naming) Use a suitable font size for the element name. Discuss

Discussion: Keeping the model neat and organized helps in producing more readable and usable models.



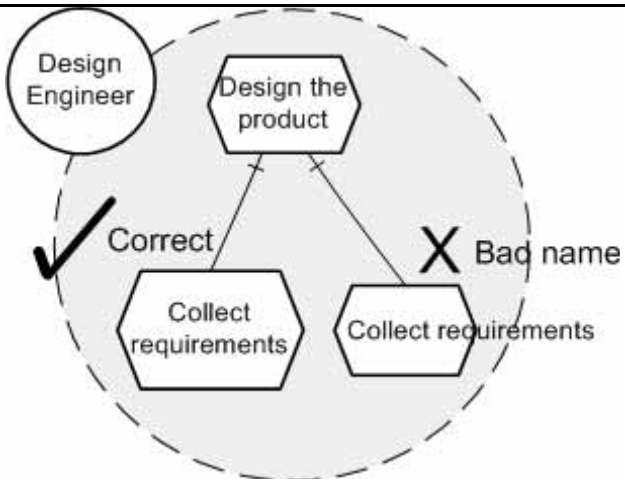
4.3.4 Guideline (Beginner,Naming) Select concise but informative phrases to name the elements. Discuss

Discussion: Softgoal naming must be precise and conveys meaningful information. For example, a Softgoal might be labeled as: Design Process Be Efficient or alternatively, Efficient Design Process. Avoid using very vague or very abstract such as: “The process is working good”. The following illustration depicts various examples of good and bad naming for various elements.



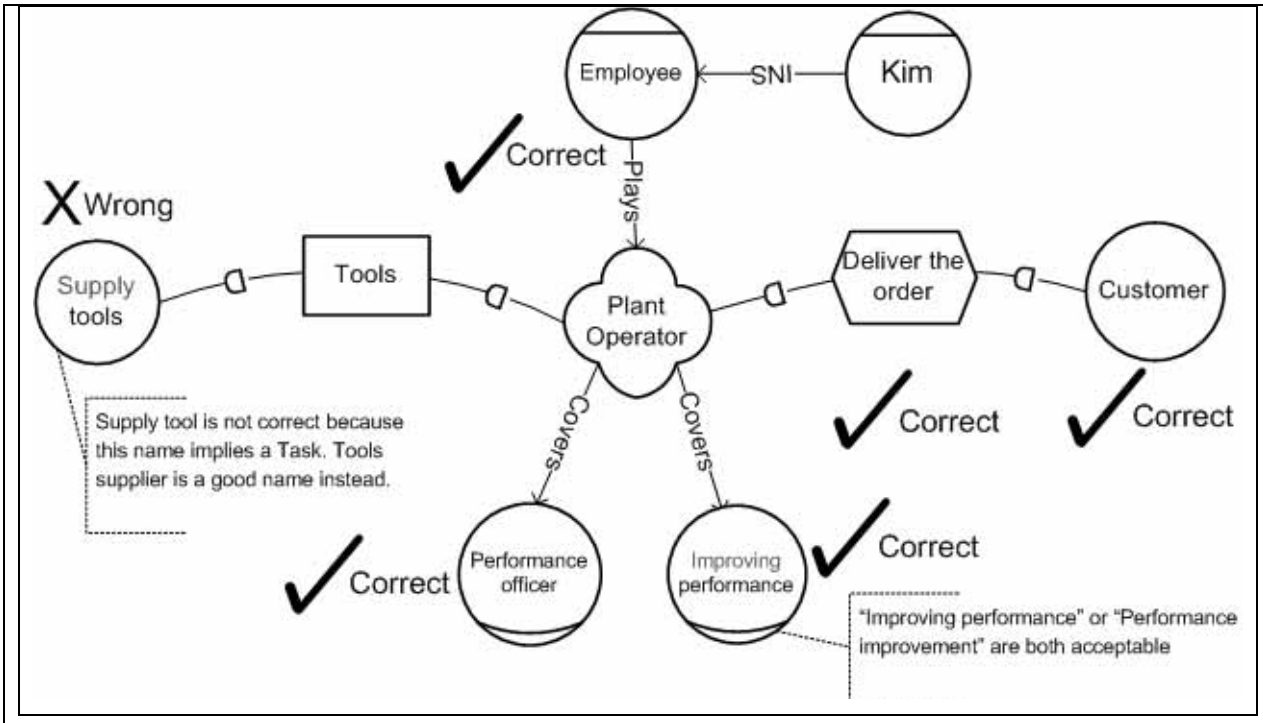
4.3.5 Guideline (Beginner,Layout) Don’t extend the text of the name of the element beyond the element’s boarder. Discuss

Discussion: Keeping the model neat and organized helps in producing more readable and useable models.



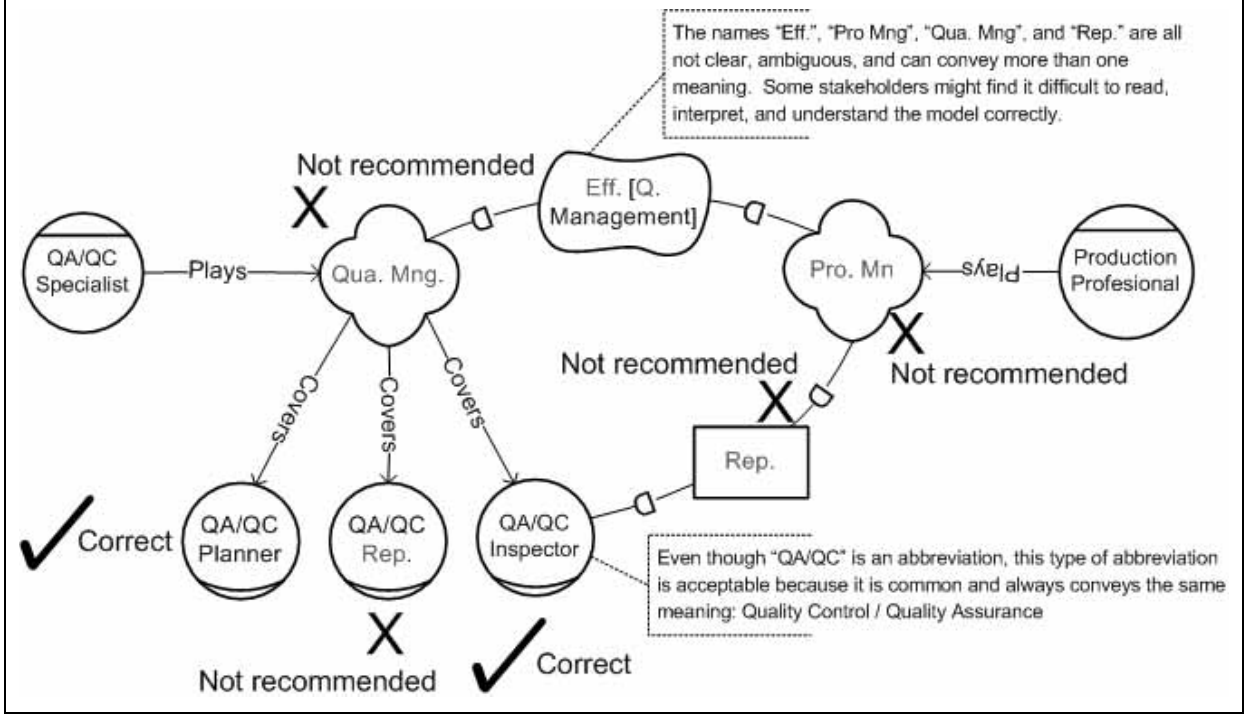
4.3.6 Guideline (Beginner,Naming) Do not use Verbs in the names of Actors, Agents and Positions. Discuss

Discussion: Names of Actors, Agents, and Positions should not be Verbs. Avoiding using Verbs in these elements helps in eliminating readability conflict with Tasks, which are described by Verbs. Agent names can use general names such as “Employee” or specific names such “Judy”. Roles are different than Tasks. Tasks are used when an actor (Depender) depends on another actor (Dependee) through the strategic Dependency Link, to accomplish a task, which has a clear idea of achieving it. The Dependee, however, still has freedom to carry out the activity within some constraints. A Task is usually is described by decomposing it into sub-elements, which could be another task, softgoal, goal, or resource. Also, Tasks don’t always have to be involved in a dependency. An actor can have internal tasks, which are not involved in a dependency. In contrast, a Role is not a Task but an abstract characterization, which is played by an Agent or Covered by a Position. Also, A Role is specific type of an Actor, which can contain its own elements such Goals, Softgoals, Tasks, and Resources. Therefore, it should not be confused with Tasks, which are just one possible element of a Role. Also, not only human Actors can play Roles, but also nun-humans (such as systems) can play Roles.



4.3.7 Guideline (Beginner,Naming) Use clear names without ambiguous and unknown abbreviations or acronyms. Discuss

Discussion: Names of Actors, Agents, Roles, and Positions should be clear. Ambiguous abbreviations might make it harder for some of the stakeholders to read, interpret, and understand the model. Therefore, abbreviations are not recommended unless a legend is supplied to explain their meaning.



4.4 Scaling

Models could grow in terms of the number of Actors of a model, number of external and internal elements that are associated with these Actors, the level of the refinement and decomposition of these elements, and the depth of the concepts that represent the domain of the system that is being modeled.
When models grow and become more complex, scalability issues may arise. Therefore, using some guidelines that deal with these issues could help in suggesting ways of presenting complex models to the users of the models in a readable, easy to use, and usable manner. This section provides some guidelines on scaling and complexity issues.

4.4.1 Guideline (Intermediate,Layout) Split a large and complex model into consistent pieces to facilitate easier presentation and rendering. Discuss

Discussion: Large models may be split into consistent pieces to facilitate easier presentation and rendering. In such instance, the model's pieces need to be consistent with each other. Each piece is just a view or subset of a larger conceptual model, even if a physical representation of this model doesn't exist due to space constraints. As shown in the illustrations extracted from Horkoff, Yu, and Liu (2006), Figures 2 to 4 are pieces or segments of the complete SD and SR diagrams in Figures 1 and 5. The objective of presenting this example here is not to go through the internal logic or content of the case, but to illustrate the overall layout and arrangement of consistent pieces or segments of complete SR and SR models. Please refer to "Trusted Computing: An i* Case Study" by Horkoff, Yu, and Liu (2006) for full details about this case.

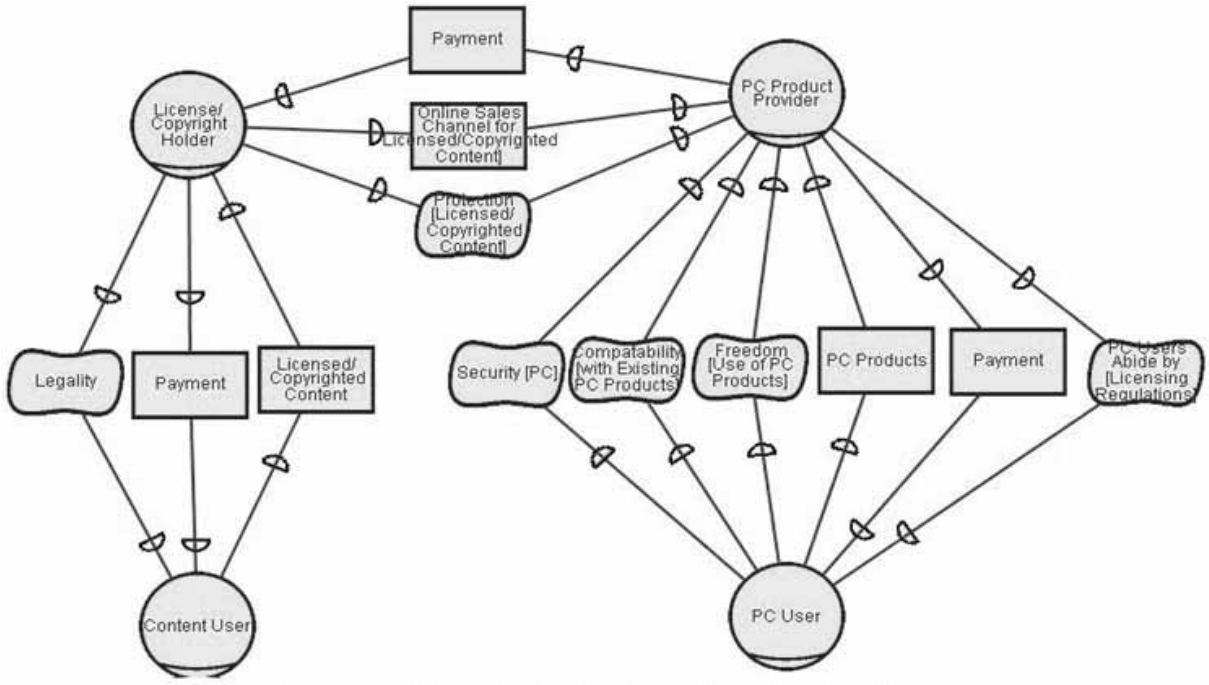


Figure 1 The complete SD i* model of Trust Computing (TC) case study

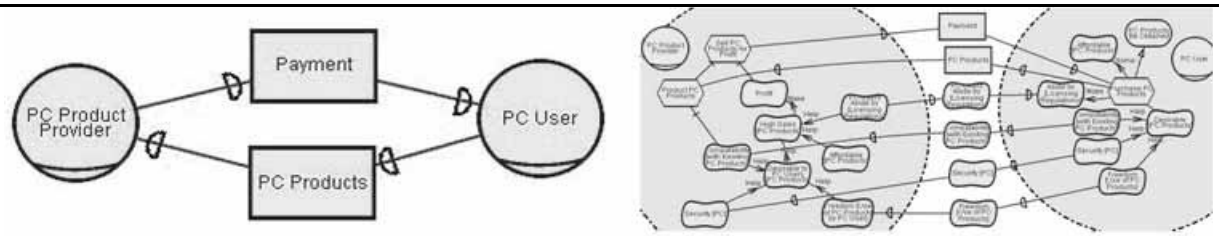


Figure 2 A model segments showing two Actors of the model

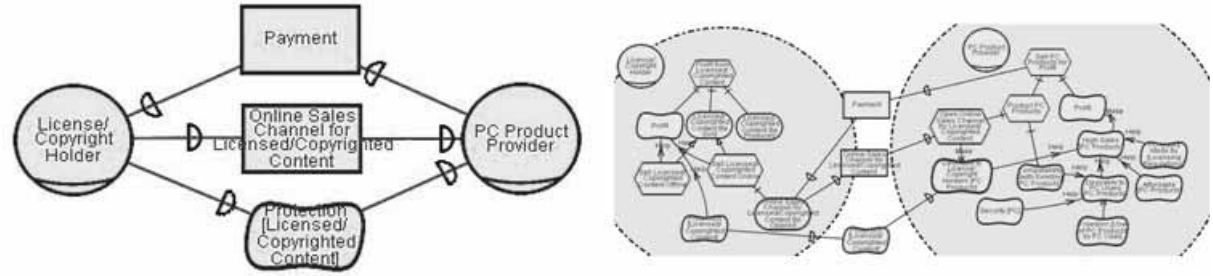


Figure 3 A model segments showing another two Actors of the model

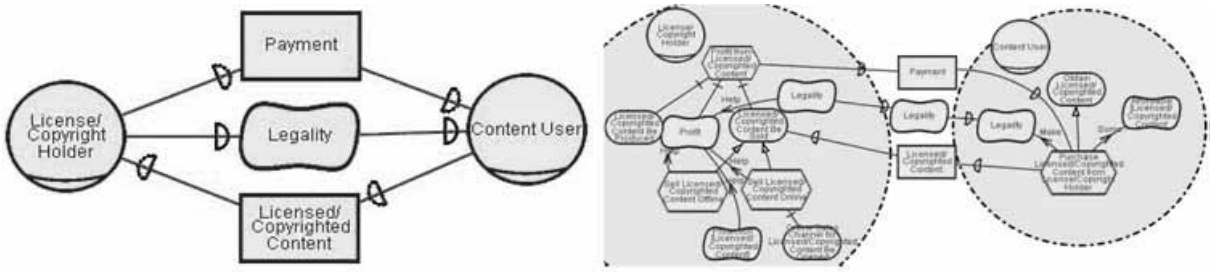


Figure 4 A model segments showing another two Actors of the model

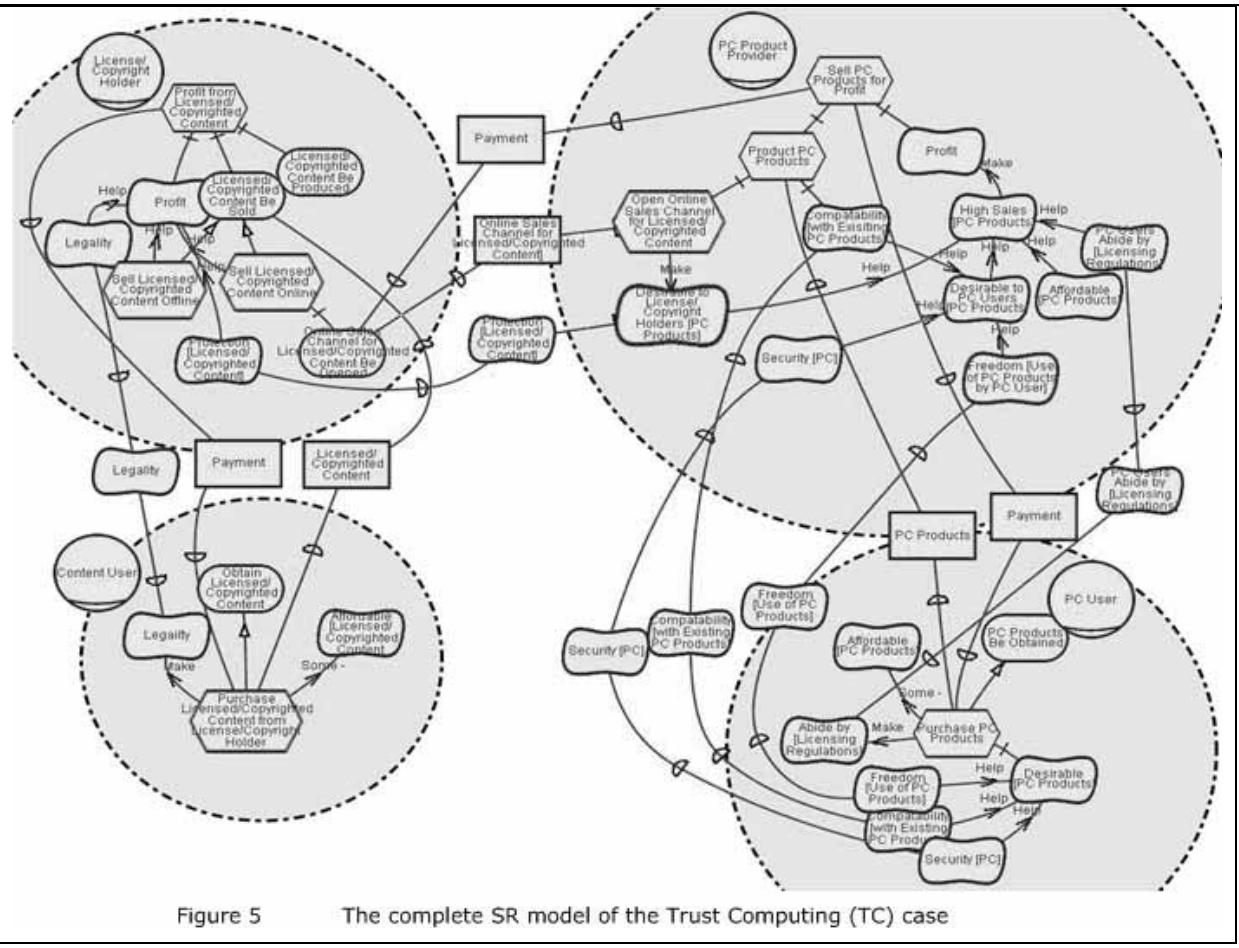
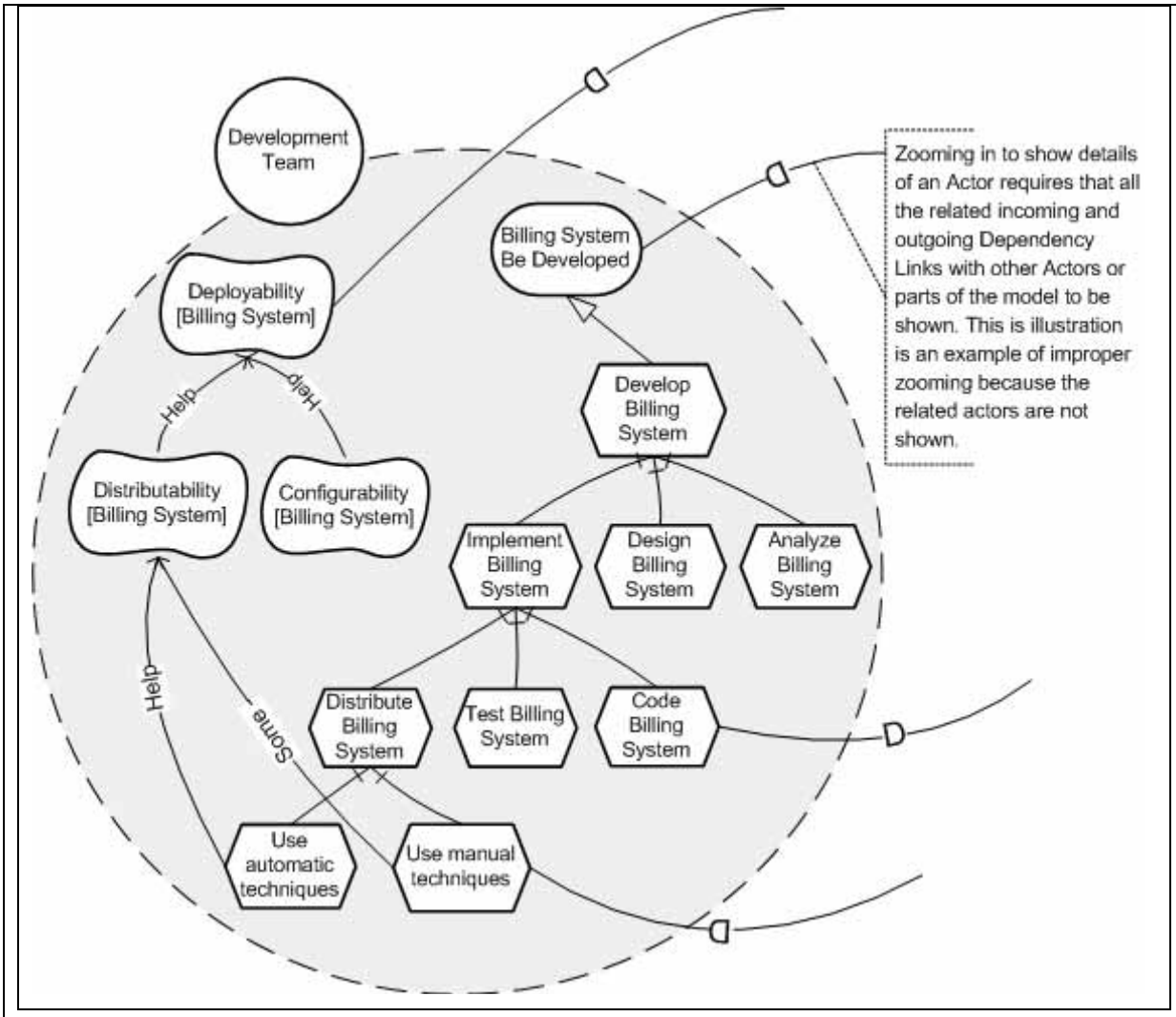


Figure 5 The complete SR model of the Trust Computing (TC) case

4.4.2 Guideline (Intermediate,Layout) Don't extract or zoom into a section of an Actor in a model without showing the incoming and outgoing dependencies with other Actors or parts of the model. Discuss

Discussion: Zooming into or extracting a section of a model is one way to facilitate working and analyzing sections or parts of a large or complex model. Zooming, however, need to be dealt with cautiously. All the related incoming and outgoing links of the zoomed section need to be shown. Neglecting or missing some of the links leads to inconsistent model, which can be hard to read, interpret, understand, and communicate.



4.5 Level of complexity

4.5.1 Agents, Roles, and Positions

4.5.1.1 Guideline (Intermediate,Layout) **Use the specialized actor notation to the degree that you can gain advantage and higher level detailing in instantiating the actual stakeholders.** **Discuss**

Discussion: It has been outlined in Section 1.3.1.1 “Actors” the guideline of using Agents and Roles instead of the general Actor notation when the distinction is easily made. Choosing to use the specialized Actor notations such as Agent, Roles, and Positions can help in gaining higher level of detailing in instantiating the actual stakeholders and capturing the knowledge domain. On the one hand, lack of use of any of these specialized actor notations might subject the model to some lose of useful information. On the pother hand, excessive use of the special actor notations might lead to much more complex models that might become harder to deal with. Therefore, it is recommended

to subject the choice for the use of the general Actor versus the specialized Actor notation based on the value and additional information that they will add to the model.

4.6 Model Analysis

The i* Framework provides concepts which facilitate the analysis of the success of domain situations represented in models.

4.6.1 Ability

When an agent has a routine that can serve some purpose, e.g., to achieve a goal, we say that the agent has an ability to achieve that goal. In this formulation, an actor has the ability to do something if it has a routine for it. Having ability does not necessarily imply that one can achieve something all by oneself. Various degrees of delegation and external dependency may be involved.

4.6.2 Workability

The notion of “workability” indicates that an agent believes that some routine would work (at run-time) even though it is incompletely specified or known. Thus, the actor is confident that, at execution time, it will be able to carry out the reasoning and actions required to achieve the result. A routine is judged to be workable if each of its explicitly mentioned elements is workable, and if all the constraints in the routine are expected to hold. Thus, an element is made workable by reducing it thought the routine to primitively workable elements (though not necessarily primitively executable actions), or by delegation some of the elements to other agents. A primitively workable element is one that is judged to be workable without further reduction. At an actor boundary, an open element is workable if there is some actor offering this element. A committed element is workable if there is another actor committed to producing this element as dependum to the first actor.

4.6.3 Viability

The notion of “viability” indicates what the level of evidence that indicates that the routine will work. We say that a routine is viable if all its softgoals are satisfied. Softgoals at the top level are selectively applicable to elements at lower levels in the functional means-ends hierarchy, as selected by the means-ends hierarchy of softgoals, via the parameters in the softgoal nodes.

4.6.4 Believability

Since the SR model relies on judgment and argumentation, there are many assumptions made as the model is constructed. A fourth level of assessment is in the believability of these assumptions. A qualitative treatment of these is also assumed.

4.6.5 Evaluation and Propagation

This section discusses the main concepts, notation, and qualitative process or procedure of using i*

models for evaluation. General recommendations and guidelines, associated with discussions and illustrations, are provided as well. It is intended in this i* Guide to present the fundamental concepts and the process in a simple and easy to understand manner. Is it a good staring point to explore i* evaluation. The intent of this section is to bring about some of the necessary elements (such as propagation rules) of conducting a sound analysis to arrive at reasonable answers to the analysis questions. For additional discussions on using i* models for evaluation, including topics of Evaluation script and algorithm, convergence, termination, human judgment, and giving labels to non-leaf elements, we would like to refer you to the original references upon which the content of this section has been based. These references are: Horkoff (2006), Horkoff (2004), Horkoff and Yu (2005), Horkoff, Yu, and Liu (2004), Horkoff, Yu, and Liu (200), and Horkoff and Yu (2005). Please refer to the reference section at the end of the Guide for full citations of these resources. Also, the example that is used in Evaluation Step 2 in Section 4.6.5.1.5.2 does not depict the underlying algorithm for the Evaluation procedure step-by-step, but combines simpler steps together to give more condensed example

This i* Guide has provided in various sections some recommendations on the use of i* notation for developing consistent, readable, understandable, and usable SD and SR models. Also, SR models and their associated elements have been discussed in section 4.2 “Strategic Rational (SR) Model”. An SR model is the place where the modeler works within Actors to model their internal elements and model the way these elements are thought to contribute to the satisfaction and fulfillment of Actor’s internal goals and other Actors’ dependencies via the Dependency links.

Now, the question is how can the modeler tell whether the model and its web of internal elements really fulfilled and satisfied Actors’ goals and dependencies? What options are available? Or what is the effect of one element on other elements? The answers to these questions touch the very essence of i* modeling framework, which uncovers the reasoning and rational behind making informed modeling decisions through using i* models for evaluation. Also, i* evaluation is believed to have another important advantage of enhancing the overall quality (accuracy and comprehensibility) of i* models by prompting the modeler for change as a result of verifying the syntax and validating the meaning of the models.

As we will see in this section, human intervention or judgment arises as a useful or maybe even a necessary strategy in the evaluation process, especially when modeling the complex interactions of an organizational network. Therefore, because it is infeasible to create a model which captures the complex interactions completely, informed and careful human intervention in the evaluation process is allowed to compensate for some issues related to lack of information or model precision, correctness, and completeness. The “Propagate Label Values” step of the evaluation process presented below provides additional discussion about this topic.

4.6.5.1 Evaluation Procedure

The following is the qualitative procedure that is recommended to be followed to conduct an evaluation

using i* models. To illustrate the procedure in a practical way, a series of simple models, which are extracted from Horkoff (2006) is provided. To facilitate the understandability and comprehensibility of the procedure, each model is associated with a discussion that points to the evaluation step and label propagation rule being used. The referenced rules are provided only as guidelines that can be adapted by intermediate and advanced users as appropriate in more advanced situations. The ultimate objective of this section is to provide the modeler with a method that can help in producing understandable, comprehensible, and transferable models. One way to achieve this objective is by utilizing the recommend steps to systematically develop models that are characterized by more explicit facts than tacit knowledge in the minds of the modelers and evaluators. The following sub-sections discuss in detail all what the evaluator needs to get started.

Evaluation procedure or process consists of several steps, which are listed below. Each step is discussed in sufficient detail in the subsequent sub-sections to get started with the procedure.

- Step 1 Decide what analysis question you are asking the model
- Step 2 Give initial Labels to “leaf nodes or elements” based on the question that you have decided to apply in Step 1
- Step 3 Propagate Label values
- Step 4 Interpret the results
- Step 5 Repeat steps 1 to 4 for each analysis question

4.6.5.1.1 Guideline (Intermediate,Evaluation) **To achieve more reliable and accurate evaluation results, employ a systematic evaluation procedure instead of using a manual mental method to guess the answer for an analysis question. Discuss**

Discussion: The modeler might attempt to use a mental or manual method of guessing the answer for an analysis question by tracing through the links without going through the systematic evaluation procedure that we are discussing in this section. While this method can produce some analysis results for relatively simple models, it is highly likely that evaluating more complex models can produce analysis results that are difficult to acquire by manually examining the model. Therefore, the benefits of obtaining more accurate and reliable analysis results justify and pays off spending the required time to evaluate i* models.

4.6.5.1.2 Guideline (Intermediate,Evaluation) **In order to fully account the effects of all elements in the model, follow the evaluation procedure and steps in order. Discuss**

Discussion: The evaluation procedure discussed in this section provides the modeler with a step by step and systematic approach to the logic and rational behind the evaluation process. The intent of this procedure is to bring about some of the necessary elements (such as propagation rules) of conducting a sound analysis to arrive at reasonable answers to the analysis questions. See Horkoff (2006) for a fuller and more advanced and detailed discussion, including Evaluation script and algorithm, convergence, termination, human judgment, and giving labels to non-leaf elements.

4.6.5.1.3 Evaluation Step 1: decide what analysis question you are asking the model

This is the first step, or you may call it the pre-requisite step for the first step, of the evaluation process. At this point, the modeler poses a question for the model to answer. Multiple questions could be formulated to explore and compare various "what-if" scenarios. Also, one "what-if" question may involve multiple parts so the modeler can test the effects of more than one element on more than one element.

Figure 2 in the next section 4.6.5.1.4 depicts one model out of a series of relatively simple models that are created to illustrate the evaluation procedure. At this stage, we would like to pose an analysis question to the model and try to use the model to answer this question. The question that we would like to explore is: If the PC product Provider (Role) decides to not Allow Peer-to-Peer Technology (Softgoal), what effect will this have on Sell PC Products for Profit (Task)? The next steps of the evaluation procedure walk through the model to answer this question.

Other questions could be asked. For example: if the PC Product Provider (Role) decides to Allow Peer-to-Peer Technology (Softgoal), what effect will this have on Affordable Pc Product (Softgoal) of the PC User (Role)? Also, if the PC User (Role) decides to Obtain PC Products from Data Pirates (task), what effect will this have on the Profit (Softgoal) of the PC Product Provider (Role)? The Modeler might answer a question about elements that have simple and noticeable interactions. Generally, an answer to a question tends to be harder when the relationship between the nodes must be traced though the model.

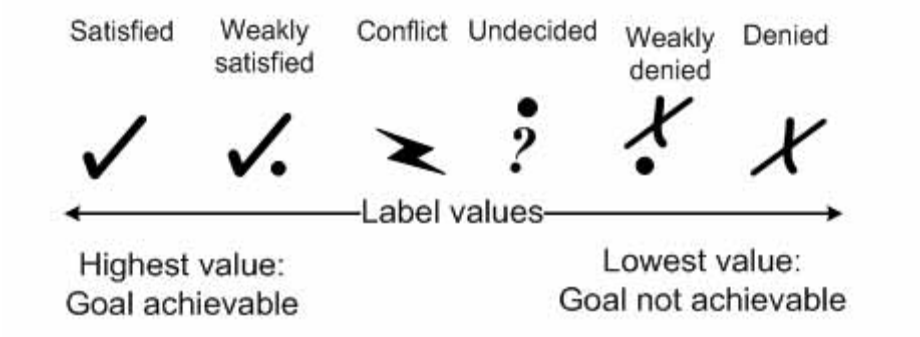
4.6.5.1.3.1 Guideline (Intermediate,Evaluation) Formulate an analysis question before giving initial labels to leaf nodes or elements. Discuss

Discussion: Step 1 of the Evaluation procedure is a significant step because it orients the modeler to give the initial evaluation labels as discussed in the Steps 2 of the Evaluation procedure. Overlooking this step and jumping to the next one is analogous to inquiring a search engine without supplying the search criteria.

4.6.5.1.4 Evaluation Step 2: Give initial Labels to "leaf nodes or elements" based on the question that you have decided to apply in Step 1.

Leaf elements are the starting points in the evaluation. They are not decomposed into further elements and have no input links because they are the lowest level in the refinement or decomposition hierarchy within Actors. In i*, leaf elements could be soft elements such as Softgoals, or hard elements such as Goals, Tasks, and Resources. The possible initial label values that can be assigned to leaf elements are illustrated in Figure 2. As depicted in the

Figure 2, values range from the lowest or minimum (Denied) to the highest or maximum (Satisfied) with partial and other values in between.



These labels represent qualitative judgment of the level of the elements' satisfaction or denial. Table 1 provides explanations of these labels:

Table 1 Explanation of node or element labels. Source: Horkoff and Yu (2005)

Node or element Label	Explanation
Satisfied	When the developer considers that the achievement of a softgoal is satisfactory, the softgoal is satisficed
Partially/Weakly Satisfied	Representing inconclusive positive support for a parent.
Conflict	If it is both satisficeable and deniable
Undecided	If it is neither satisficeable nor deniable (default label).
Partially/weakly denied	Representing inconclusive negative support for a parent
Denied	When softgoal achievement is considered unsatisfactory, the softgoal is denied

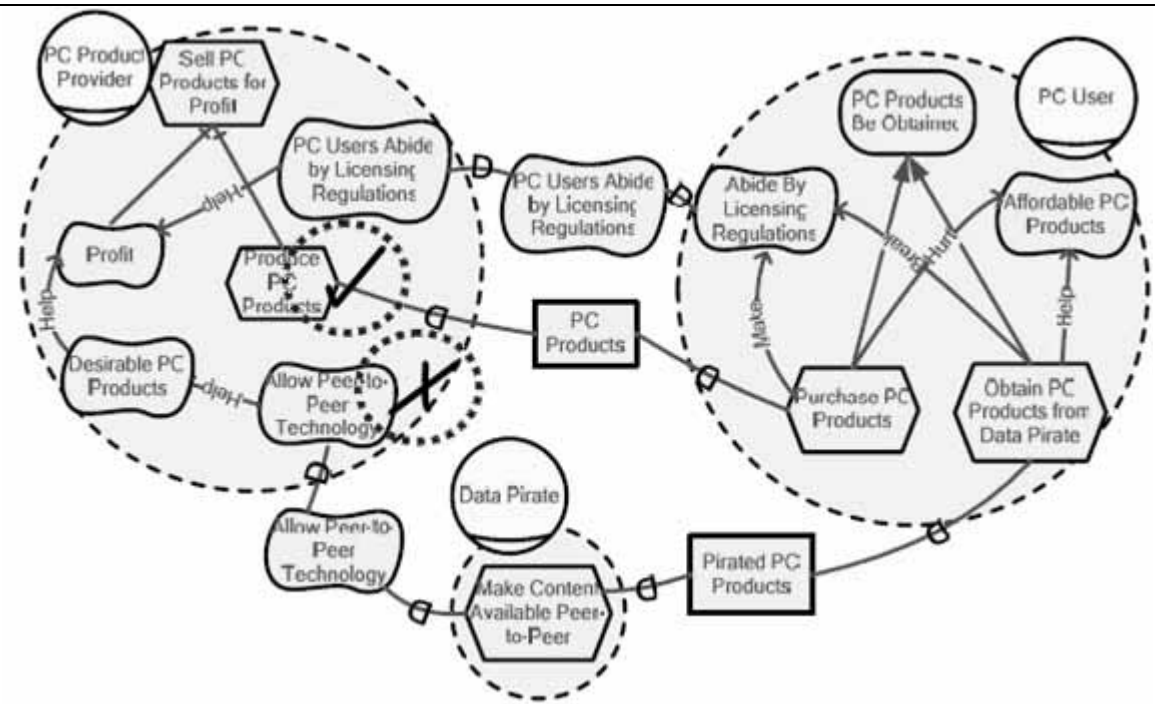
Section 4.2.2 "Element/Nodes" explains the meaning of and the difference between i* elements. Please refer to section 4.2.5 "Contribution Links" for discussions about various types of Contribution Links. Table 2 provides explanations of various Contribution Link Types:

Table 2 Explanation of Contribution Link Types . Source: Horkoff and Yu (2005)

Contribution Link Type	Explanation
Make	A positive contribution strong enough to satisfice a node
Some+	Either a make or a help contribution, a positive contribution whose strength is unknown
Help	A partial positive contribution, not sufficient by itself to satisfice a node
Unknown	An unknown contribution to a node
Or	The parent is satisficed if any of the offspring are satisficed
And	The parent is satisficed if all of the offspring are satisficed
Equal	The contribution to the node is equal to the label of the node making the contribution
Hurt Some-	A partial negative contribution Either a break or a hurt contribution, a negative contribution whose strength is unknown
Break	A negative contribution sufficient enough to deny a node

For example, **Figure 2** depicts an illustration of giving initial labels that correspond to the analysis question to two leaf elements, which are circled in red.
We marked the Allow Peer-to-Peer Technology (Softgoal) as Denied and marked Produce PC Products (Task) as Satisfied to get started with answering the question stated earlier.

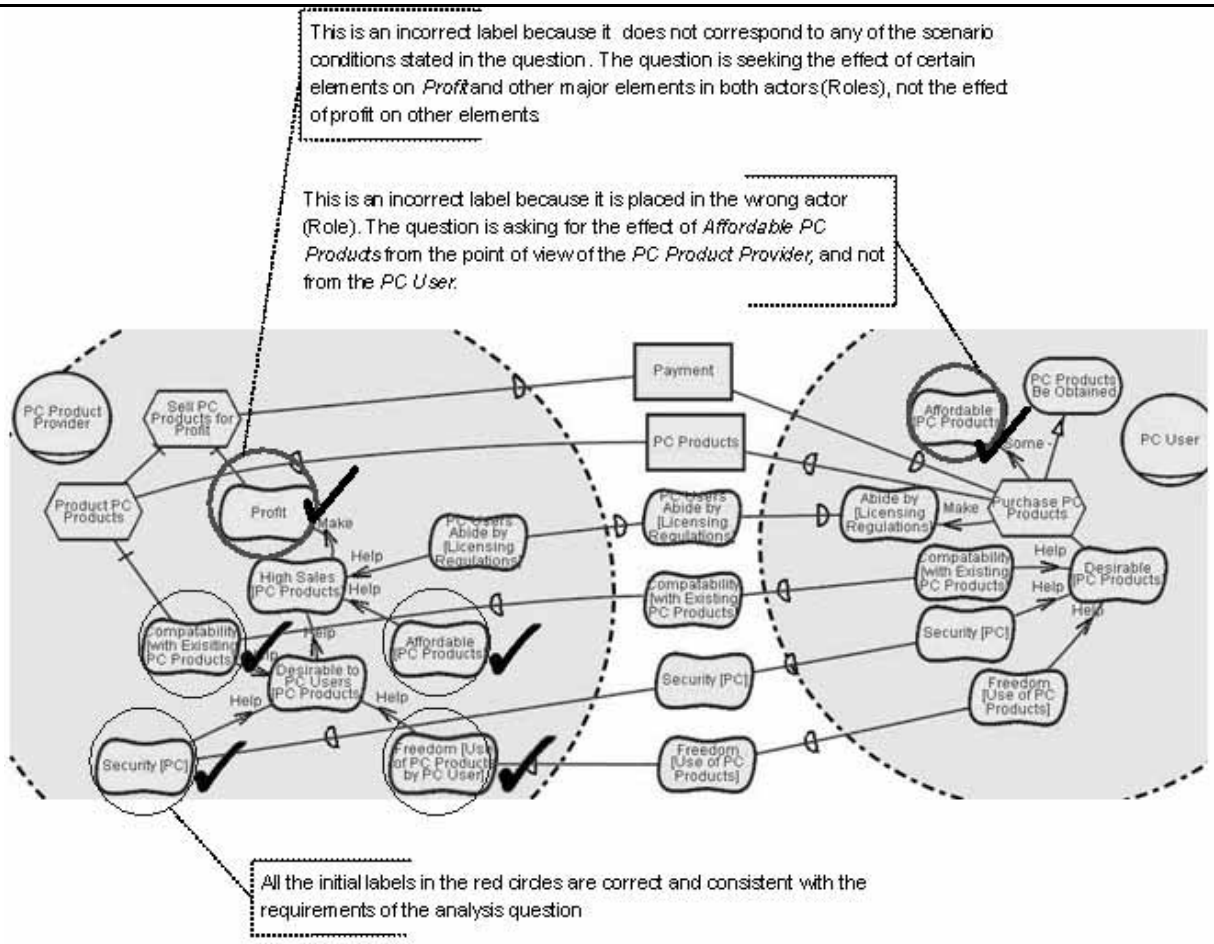
You can also see in **Figure 2** the labels of the Contribution Links, which were used as an integral part of developing the SR model itself. We can see, for example, Purchase PC Product (task) in PC User (Role) has two Contribution Links: The Make Contribution Link to Abide By Licensing Regulations (Softgoal) and the Hurt Contribution Link to Affordable PC Products (Softgoal). You can also see other Contribution Links used in this sample model in both PC User and PC Product Provider Roles. There are no Contribution Links that are used or shown in Data Pirate (Role) in this simplified illustration.



4.6.5.1.4.1 Guideline (Intermediate,Evaluation) **Give initial labels to the leaf elements in a consistent manner with the analysis question or scenario. Discuss**

Discussion: The purpose of labeling or highlighting the leaf elements is to mark those elements that correspond to the analysis question and work as a guide to the starting point from where the evaluation and propagation starts. So, this labeling activity makes it easier for the readers of the model to understand what initial leaf elements were considered for answering the analysis question. Let us look at the example in the following model. Suppose the analysis question is: "If PC Products which meet the PC Product Provider's standards for Compatibility, Security, Freedom of Use and Affordability are produced, are the main elements of the PC User and PC Product Provider satisfied?" The illustration depicts a correct set of initial labels for the leaf elements (thin red circles) that correspond to the analysis question and an incorrect set of labels (thick blue circles).

It is also often necessary to give initial labels to leaf elements that are not necessarily directly related in the analysis question in order to have a propagation that that makes sense. In Figure 2 presented in Evaluation Step 2, Produce PC Products is an example. The molder doesn't really care that much if actors produce products, it's just a necessary leaf node that needs to be given a value in order to see the results of not allowing peer to peer technology.



Source: Trusted Computing: An i* Case Study. (Horkoff, Yu, and Liu, 2006)

4.6.5.1.5 Evaluation Step 3: Propagate Label Values

The next step in the qualitative evaluation procedure is to propagate the labels that are given by the modeler to the leaf elements to other nodes. Label propagation is based on a set of propagation rules, used with human judgment, to determine the resulting label.

The following sub-sections discuss label propagation rules, the underlying algorithm, and give an example of applying these propagation rules. The propagation rules are provided only as guidelines, which can be adapted by intermediate and advanced users as appropriate in more advanced situations.

4.6.5.1.5.1 Label Propagation Rules

Label propagation rules that deal with Dependency Links, Contribution Links, Decomposition Links, and Means-Ends Links are classified and summarized in the following list of 10 rules:

Dependency Links related rules:

1. A Dependum acquires the value of the Dependee.
2. The Dependers take on the value of the Dependum.

Decomposition Links related rules:

3. The Decomposition Link relationship is treated as an “and” in the evaluation process and therefore all of the decomposed elements must be satisfied in order for the upper level task to be satisficed. With the “and” logic, the upper level task acquires the minimum value given or marked to lower level elements. Please refer to Figure 1 in Evaluation Step 2 above for the possible labels values.

4. If a node is involved in decomposition as well as a dependency, the values produced by these two relationships are resolved using “and” logic and therefore the node acquires the minimum or lowest value.

Means-Ends related rules:

5. The means-ends relationship is evaluated using “or” logic between the means. With the “or” logic, the Ends take on the maximum value given or marked to the Means.

Contribution Links related rules:

6. The upper Softgoal that has multiple “And Contribution Links” takes on the minimum label of the contributing nodes or elements

7. The upper Softgoal that has multiple “Or Contribution Links” takes on the maximum label of the contributing nodes or elements

8. Use the rules in Table 3 to propagate the labels of multiple Contribution Links (Make, Help, Some+, Break, Hurt, Some-, and Unknown) from the lower contributing elements to the upper Softgoal.

Table 3 Propagation Rules for Contribution Links. Source: Horkoff, Yu, and Liu (2006)

Originating Label		**Contribution Link Type**						
Label	Name	Make	Break	Help	Hurt	Some+	Some-	Unknown
✓	Satisficed	✓	✗	✓.	✗.	✓.	✗.	?.
✓.	Partially Satisficed	✓.	✗.	✓.	✗.	✓.	✗.	?.
↯	Conflict	↯	↯	↯	↯	↯	↯	?.
?.	Unknown	?.	?.	?.	?.	?.	?.	?.
✗.	Partially Denied	✗.	✓.	✗.	✓.	✗.	✓.	?.
✗	Denied	✗	✓.	✗.	✓.	✗.	✓.	?.

9. In cases when there is only one label, or when combining full and partial value or evidence, the final label for an element can be determined automatically. For example: the final label of an element receiving a bag of labels {Partially Satisfied, Satisfied} can be set automatically to Partially Satisfied. Table 4 lists the cases that can be resolved automatically.

Table 4 Cases where Final Labels for Parent Goals can be Automatically Determined. Source: Horkoff (2006)

Case	Resulting Label
1. If the bag has only one label	the single label
2. If the parent goal has multiple full labels of the same polarity, and no other labels, such as {✓, ✓, ✓} or {✗, ✗}	the full label
3. If all labels in the bag are of the same polarity, and a full label exists in the set of labels, such as {✓+, ✓, ✓+} or {✗, ✗+}	the full label
4. If the previous human judgment produced ✓ or ✗, and a new contribution is of the same polarity	the full label

10. Use human judgment when dealing with a Softgoal that receives multiple labels which cannot be resolved in automatic cases. For instance, when an element receives both positive and negative values or evidence, such as {Partially Satisfied, Partially Denied}, or when multiple sources of only positive or only negative partial values or evidence is present, such as {Partially Denied, Partially Denied} and {Partially Satisfied, Partially Satisfied}, the evaluator can decide if this evidence is strong enough to satisfice or deny an element, fully or partially, or whether to the evidence is conflicting, combining the evidence to produce one of the evaluation labels.

4.6.5.1.5.2 Step By Step Label Propagation Rules Example

Now, we would like to do some practice and continue discussing the model that we have introduced in **Figure 2** in Evaluation Step 2 above to demonstrate how these propagation rules are actually applied via a series of models that are depicted in Figures 3 to 9 below. In **Figure 2** , two initial labels were given. Please refer to this figure again to have another look at these two initial labels before proceeding to the following figures outlining a step-by-step example for label propagation rules.

Also, the description associated with this example **does not depict the underlying algorithm for the Evaluation procedure step-by-step**, but combines simpler steps together to give more condensed example.

Figure 3 depicts the elements that are receiving new evaluation labels (circled in red) in this stage of the propagation process. Here is the explanation for each circle:

- Following Rule# 1, the Resource Dependum, *PC Products*, acquires the value of the Task, *Produce PC Products*, in the Dependee
- Following Rule# 1, the Softgoal Dependum, *Allow Peer-to-Peer Technology*, acquires the value of the Softgoal element, *Allow-Peer-to-Peer Technology*.
- Following Rule# 2, the Task element, *Make Content Available Peer-to-Peer*, in the Depender takes on the value of the Softgoal Dependum, *Allow-Peer-to-Peer technology*.
- Applying the 6th line in the table in Rule# 8, the Softgoal element, *Desirable PC Product*, receives the Partially Denied label from the Help Contribution Link

Figure 4 depicts the next illustration of the propagation. Please follow the elements circled in red. Here is the explanation for each circle:

- Applying the 5th line in the table in Rule# 8, the Softgoal element, *Profit*, receives the Partially Denied label from the Help Contribution Link. Notice that this Softgoal is involved in another Contribution Link which does not have a label yet.
- Following Rule# 1, the Resource Dependum, *Pirated PC Products*, acquires the value of the Task, *Make Content Available Peer-to-Peer*, in the Dependee
- Following Rule# 2, the Task element, *Obtain PC Products from Data Pirate*, in the Depender takes on the value of the Task Dependum, *Pirated PC Products*.

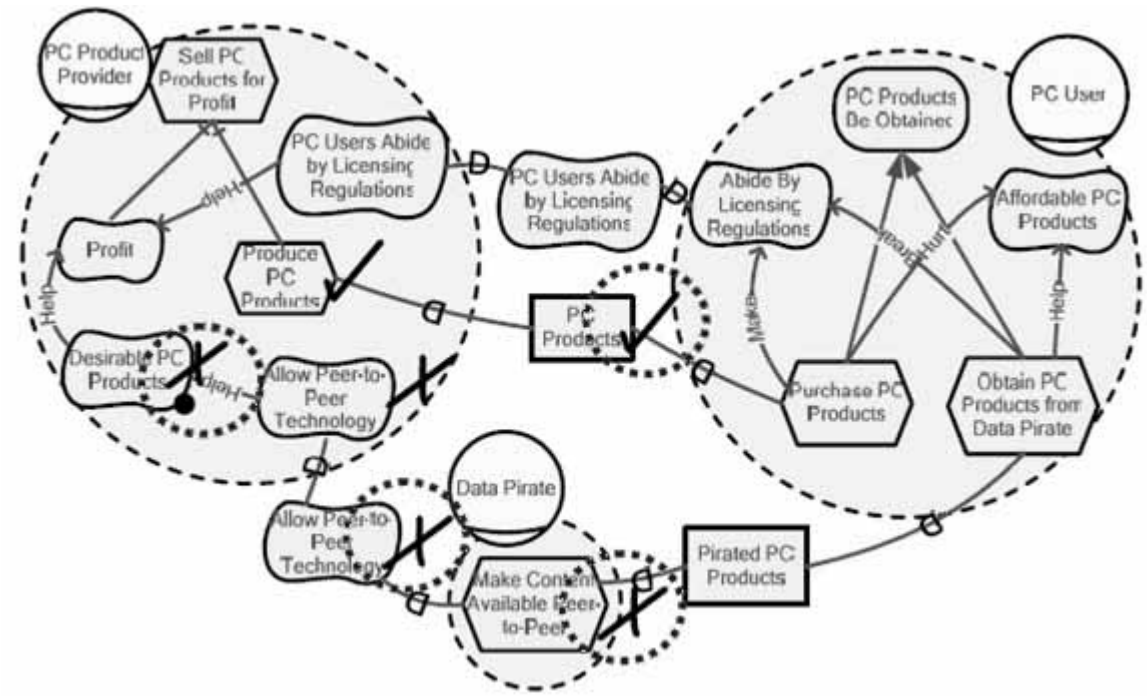


Figure 5 depicts the next illustration of the propagation. Please follow the elements circled in red. Here is the explanation for each circle:

- Following Rule# 2, the Task element, *Purchase PC Products*, in the Depender takes on the value of the Task Dependum, *PC Products*.

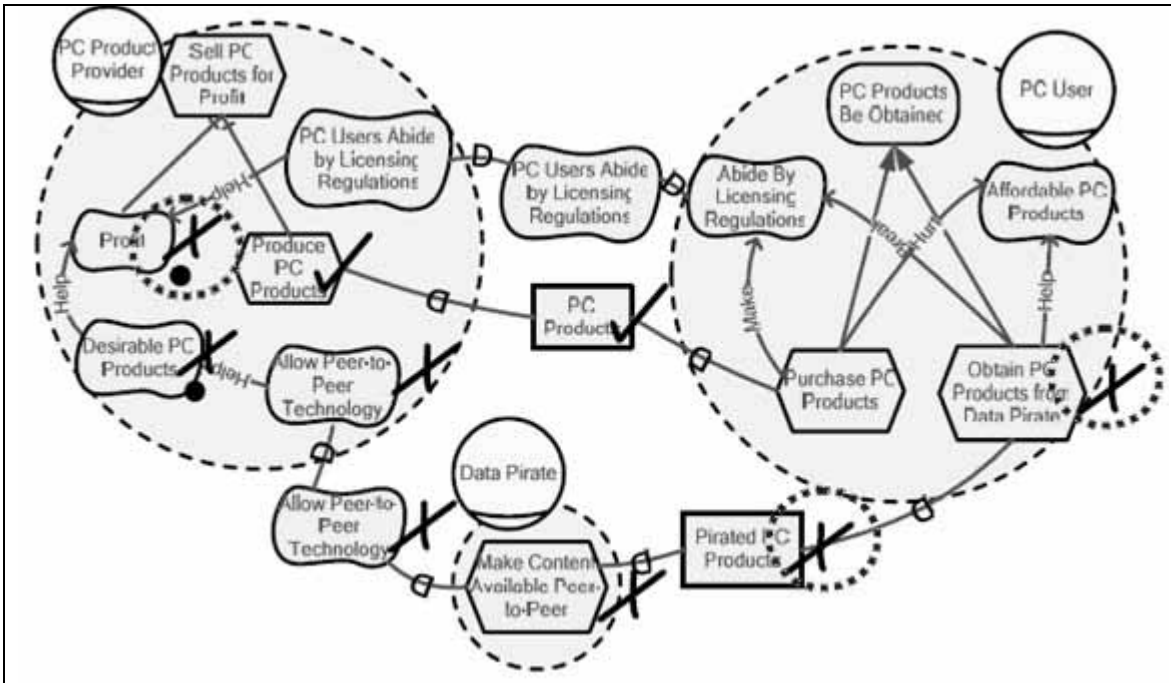


Figure 6 depicts the next illustration of the propagation. Please follow the elements circled in red. Here is the explanation for each circle:

- Following Rule# 5, the Goal element, *PC Products Be Obtained*, in the Depender takes on the maximum value of the two contributing tasks using the “or” logic.
- Following Rule# 3, the Task element*, Sell PC Products for Profit*, in *PC Products Provider*, takes on the minimum value of the decomposed elements using the “and” logic.
- Notice that the Softgoal element, *Abide by Licensing Regulations*, in *PC User* receives multiple Contribution Links from the two Tasks. Applying the first and sixth lines in the table of Rule# 8, we get a bag of multiple values: Satisficed and Partially Satisficed values. This bag is then resolved using Rule# 9 by giving the Softgoal the value or label of Satisficed.
- Notice also that the Softgoal element, *Affordable PC Products*, in *PC User* receives multiple Contribution Links from the two Tasks. Applying the first and sixth lines in the table of Rule# 8, we get a bag of multiple values: Partially Denied and another Partially Denied. This is an example of a situation where human judgment is needed. Following Rule# 10, this bag is resolved into a Denied label because the evaluator judges and decides here that there is a strong evidence to deny the Softgoal.

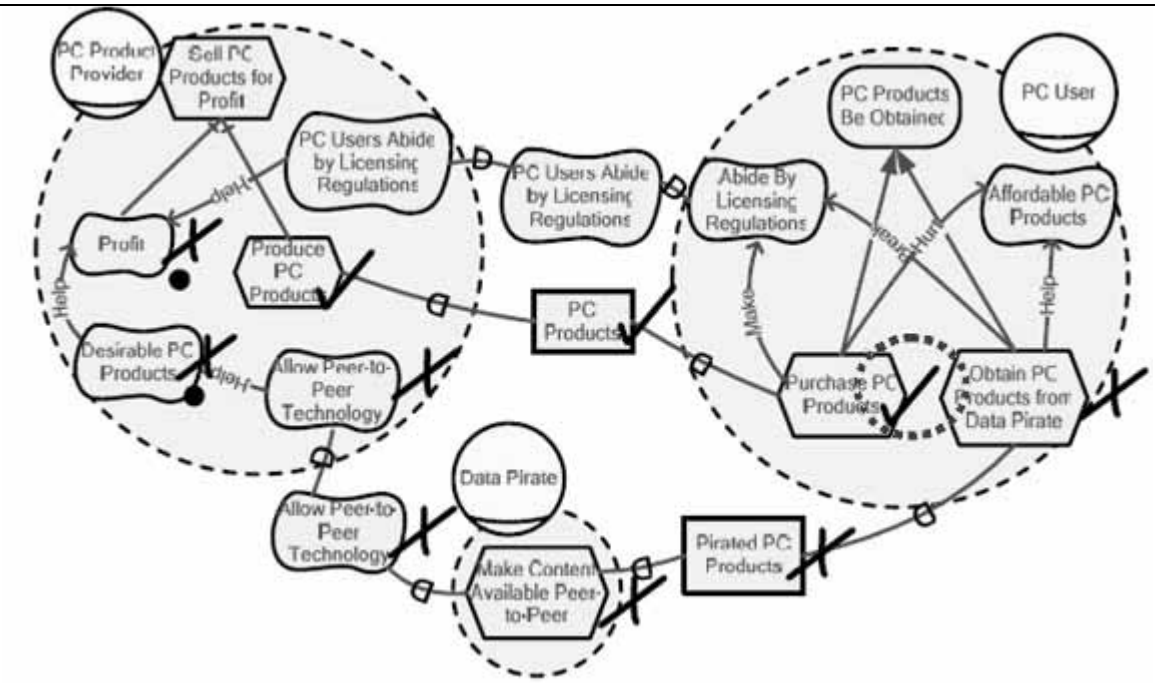


Figure 7 depicts the next illustration of the propagation. Please follow the elements circled in red. Here is the explanation for each circle:

- Following Rule#1, the Softgoal Dependum acquires the value of the Softgoal in the Dependee.
- Following Rule#2, The Softgoal element, *PC Users Abide By Licensing Regulations*, in the Depender, takes on the value of the Softgoal Dependum.

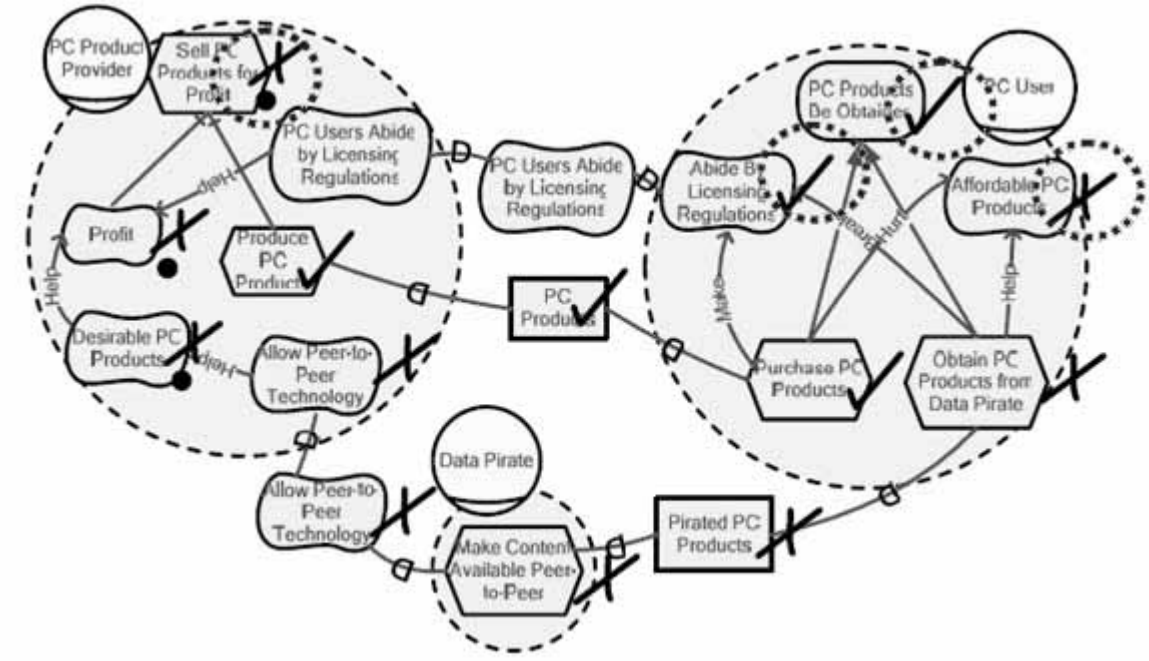


Figure 8 depicts the next illustration of the propagation. Please follow the elements circled in red. Here is the explanation for each circle:

- Notice that the Softgoal, Profit, is given the Conflict label. In Figure 5 this label was Partially Denied. Why is this happening? As you can see, the *Profit* Softgoal now has a bag of two labels from tow Contribution links. Applying the first and fifth lines of the table in Rule# 8, we get a bag of Partially Denied and Partially Satisficed. This situation is one of those that require human judgment and decision about the final label to be given to the element. The evaluator can decide that the overall evidence of the element is either positive or negative, or can decide that the evidence is of comparable strength, and give the Softgoal a Conflict evaluation label. In this example, the evaluator judged the situation and decided to give the Conflict label.

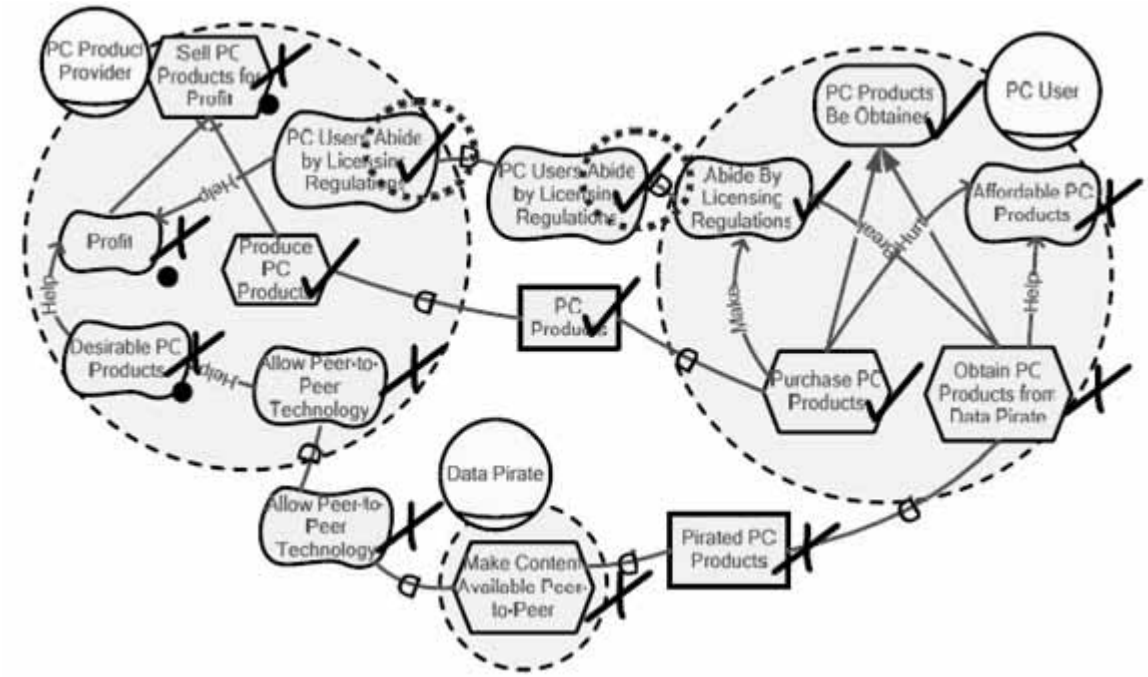
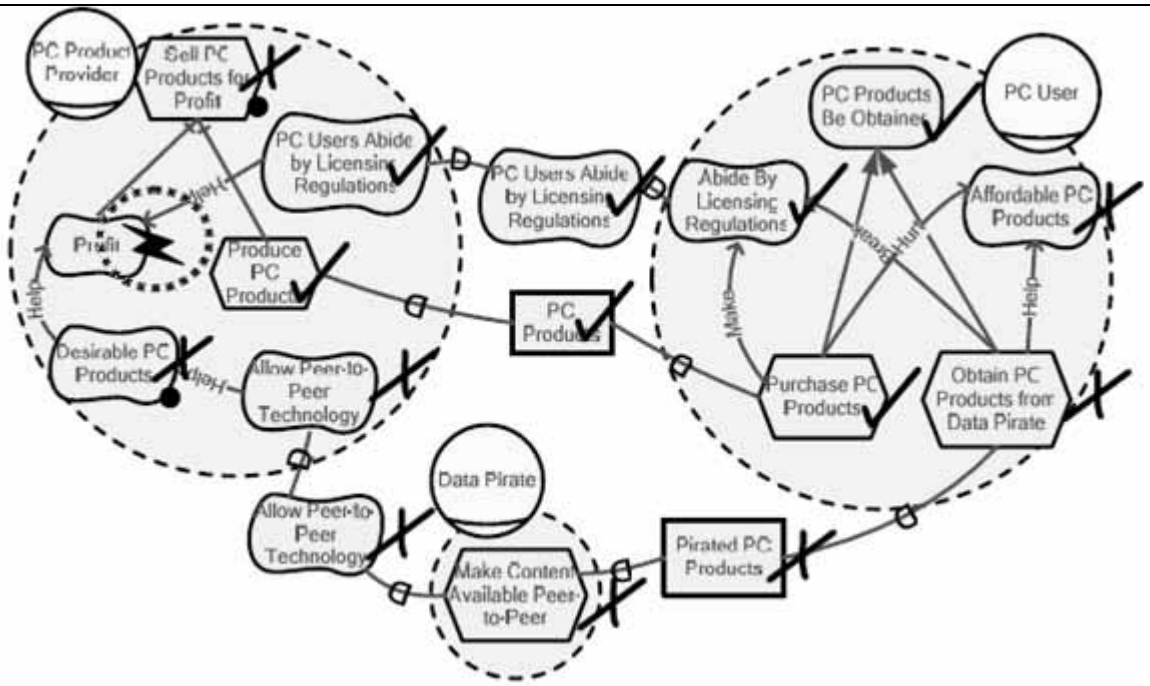


Figure 9 depicts the next illustration of the propagation. Please follow the elements circled in red. Here is the explanation for each circle:

- In this last illustration, and following Rule# 3, the “or” logic used to resolve the labels of the Decomposition Links by acquiring the minimum value. The Task element, *Sell PC Products For Profit* receives the Conflict propagation label.



4.6.5.1.5.3 Guideline (Intermediate,Evaluation) Contributions from multiple elements typically require human judgment. Discuss

Discussion: There are situations in which propagation rules follow an automatic and straight forward propagation. It has been observed, however, that in reality there are many other situations involving Contribution Links that are more complex than those suggested by the propagation rules presented in this section. In such complex situations, a soft element might receive multiple, contradictory and conflicting Contribution Links from the lower elements. In similar instances, when a soft element receives both positive and negative values (evidence), or when there is no source of sufficient evidence from lower elements, human judgment or intervention is needed to decide what label to give to the soft element. Human judgment needs to be educated and supported with utilizing the semantic meaning and the underlying contextual and domain knowledge of the elements.
Figure 5 in the step by step example illustrates similar instance..

4.6.5.1.6 Evaluation Step 4: Interpret the Results

Once the label values of the Contribution Links and the labels that were initially given to leaf elements were propagated through the model, the modeler is now ready to interpret the answer for the analysis question posed at the beginning of the qualitative evaluation procedure.

Now, in order to complete our demonstration of the ability of i* evaluation to facilitate analysis by providing answers to strategic questions, we would like to discuss the interpretation of the question that we have posed in Step 1, which is: If the *PC product Provider* (Role) decides to not *Allow Peer-to-peer*

Technology (Softgoal), what effect will this have on *Sell PC Products for Profit* (Task)?

Please refer to **Figure 9** as the reference for the following analysis. We can reason that as the *PC User* is not able to Obtain PC Products from the *Data Pirate*, in order for *PC Products to Be Obtained*, the *PC User must Purchase PC Products*. Even though this reasoning at the surface might lead to the conclusion that this situation has a positive effect on *Profit*, the final analysis result shows a conflict for Profit as a result of the *Desirable PC Products* Softgoal.

The overall result is a Conflict value for the *Sell PC Products for Profit Task*. The modeler of the example concluded that preventing the use of peer-to-peer technology will reduce piracy, but will also make products less desirable to users; therefore the overall effect on business profit for PC product providers is Conflict or both positive and negative.

4.6.5.1.7 Step 5: Repeat steps 1 to 4 for each analysis question

Once the interpretation of the results is completed, steps 1 to 4 can be repeated again to explore another analysis question with a new set of initial labels. By making multiple evaluations, the modeler can investigate several “what-if” scenarios and make informed comparisons for the strategic questions under consideration. Like with any other analytical method, as the modeler becomes familiar with the evaluation procedure, the average time and effort required to conduct future evaluations decreases and the benefit and return on investment increases and become more concrete and apparent.

Also, the level of the effort and time required to conduct an evaluation depends on the tool used in the analysis. For example, propagation process is done completely manually using istar_stencil in Microsoft Vision and done semi-automatically using OM3 tool. Semi-automatic because it allows for human judgment and intervention as discussed in the evaluation Step 3 above. Please refer to i* Wiki Website to review a variety of software tools that are available to i* modelers.

4.7 Methodology

4.7.1 Early and Late Requirements

The i* modeling framework provides the modeler with a tools-supported graphical notation that facilitates the ability to make informed choices on what procedure to follow during the modeling process. The set of guidelines discussed in this section intends to provide the beginner modeler with methodologies, or procedural choices, or approaches that can help in attaining systematic modeling processes and drawing a clear line between the As-Is situation, which captures what exists now versus the To-Be situation, which models what might exist in the future. Both AS-Is and To-Be models deal with early requirements. The To-Be model(s) can be later further decomposed to produce more detailed, later requirements.

4.7.1.1 Guideline (Beginner,Methodology) **Model the As-Is state of the knowledge domain and the current system status (if exists) without the presence of the new system To-Be introduced. Discuss**

Discussion: At this initial modeling stage, the goal of the model is to capture the As-Is situation and reflect the state of the world under analysis. The model here deals with the related stakeholders and models the current system as an actor(s) if it exists. The benefit of doing this type of analysis is to analyze why the As-Is system fails to meet the stated requirements of the stakeholders, and to form a basis for the evaluation of new system alternatives. Also, the modeler needs to state in the caption whether the model is As-Is or To-Be.

4.7.1.2 Guideline (Beginner,Methodology) **Model the To-Be state of the knowledge domain under analysis including the new To-Be system. Discuss**

Discussion: This is a more elaborate stage where the To-Be system is introduced as a new actor(s). Analyze actors, agents, roles, and positions in association with the new To-Be system to make decisions about what parts, functionalities, and requirements of the To-Be system will be done by whom and what parts of the new system. The intentional and goal-oriented capabilities, guided by the non-functional requirements, help in making decisions, which are grounded on informed and systematic rational. This can add more complexity to the As-Is model, which is used to analyze alternatives and make system design decisions.

4.7.1.3 Guideline (Beginner,Methodology) **Start the modeling with the SD model to capture the stakeholders and their associated strategic Dependency dependencies and interactions. Discuss**

Discussion: Start with this level of complexity to discover if other stakeholders have been neglected, understand the domain under analysis, and uncover any issues related to the analysis and design alternatives. SD models are relatively less complex than SR models. Therefore, this guideline could be optional and useful for students and beginners. Advanced users, however, might start with an SR model first then collapse the actors to form an SD model. Not all i* notation tools, however, can facilitate automatic collapsing for the Actors.

4.7.1.4 Guideline (Beginner,Methodology) **Employ SR models to expand on the SD models and add the intentionality and rational dimension to the analysis. Discuss**

Discussion: In case you start with and SD model to explore the various Actors in the domain under analysis, open up the Actors and systematically add new dimensions about the internal intentionality of the Actors to develop SR models that can be successfully used to uncover the rational behind the satisfaction or denial of the internal elements and external dependencies.

4.7.2 Progressive Elaboration

4.7.2.1 Guideline (Beginner,Methodology) **Start an SD model with the actors involved, then add Dependency Links starting with Resources, then Tasks, then Goals, then Softgoals.** **Discuss**

Discussion: Progressive elaboration guideline is a methodology that helps the beginner modeler to keep creating, modifying, and building upon the basic model, in an organized way to achieve the goals of the modeler. It makes SD/SR modeling more systematic as models evolve. While this guideline might be useful for students or beginner modelers, it might not be necessarily true for advanced users in all scenarios. The objective of presenting this example here is not to go through the internal logic or content of the case, but to illustrate an example of progressive elaboration. Please refer to "Montreux Jazz Festival i* Exercise on based on Osterwalder Ph.D. Thesis presentation" by Horkoff (2004) for more details about this case.

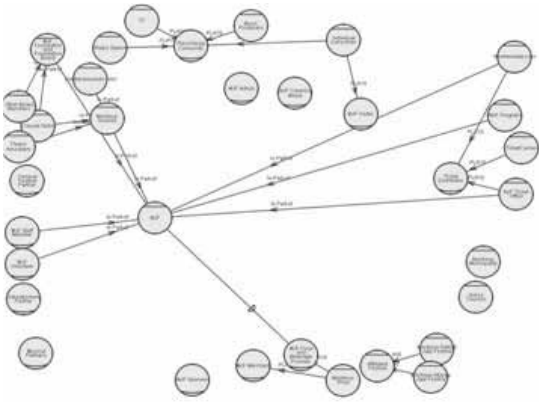


Figure 1 All Actors

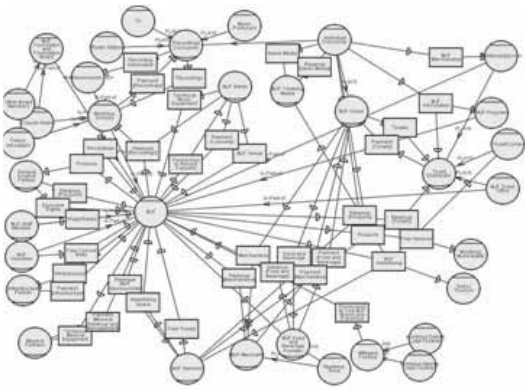


Figure 2 Add all resources

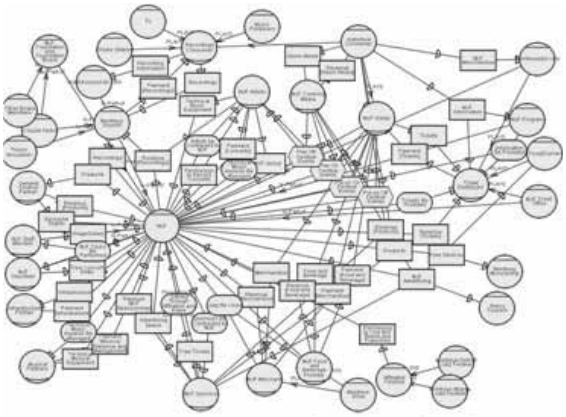


Figure 3 Everything but Softgoals

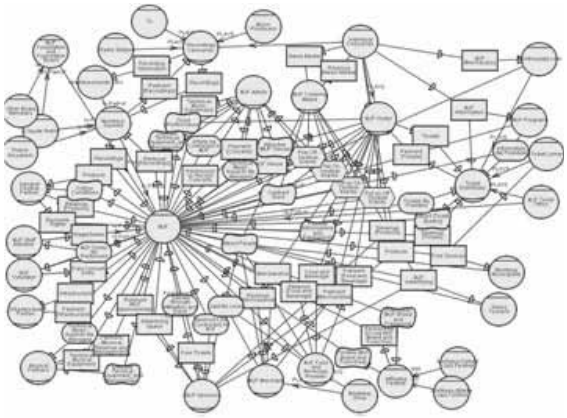
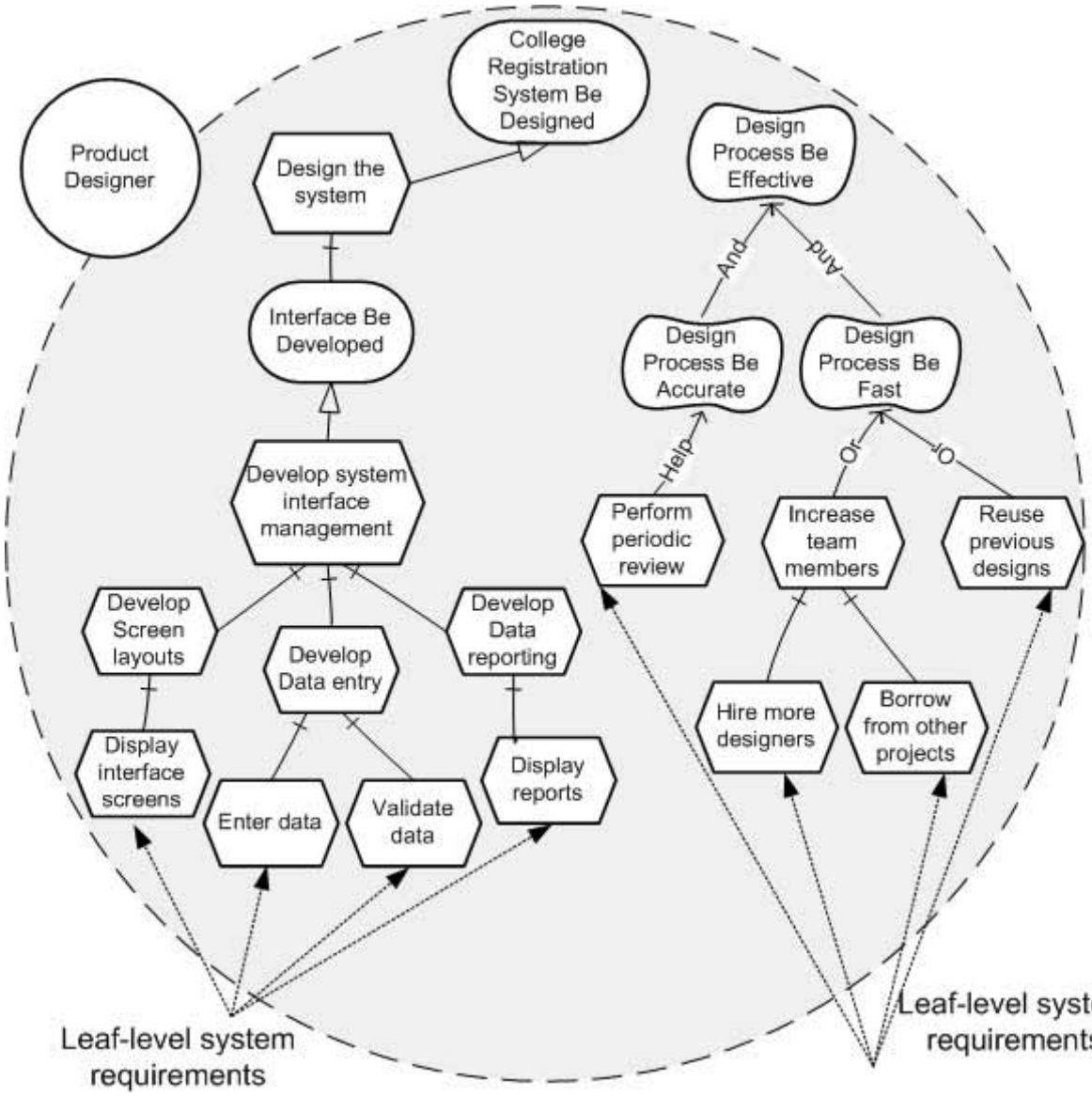


Figure 4 Big Picture

4.7.3 Guideline (Beginner,Methodology & Layout) **Use the leaf-level tasks as the system requirements, not the high level functional goals and non-functional softgoals.** **Discuss**

Discussion: High level functional goals need to be refined and non-functional softgoals need to be refined and operationalized. The Main tasks that are connected to the goals using Means-Ends Links need to be refined to lower level tasks. As well, the refined softgoals need to be operationalized to lower level tasks. All these lower level (leaf) tasks constitute the actual system requirements. The level and depth of the refinement, however, depend on some factors such as, the efforts that need to be spent in modeling, the scalability of the model, and the purpose of the model. Therefore, the

modeler might try to acquire low-level requirements, or sometimes only assess high-levels options.



5 Acknowledgments

The i* Guide owes almost everything to the numerous people who took the time to help in the preparation, review, and publishing of the Guide. Prof. Eric Yu and Jennifer Horkoff provided very helpful and critical feedback and guidance on both the content and organization of the Guide. The Guide would not have been possible without their continuous review, motivation, and encouragement.

Special thanks also go to Dominik Schmitz for his time, proactive and relentless efforts, and invaluable technical advice and assistance for publishing the Guide on i* wiki. I am also grateful to Gemma Grau for her thoughtful suggestions and ideas to improve the usability and benefit of the Guide to i* users. Thanks also to Reza Manbachi and all the anonymous individuals who thankfully contributed to publishing the Guide.

6 References

Horkoff, J. (2006). Using i* modeling for evaluation, Master Thesis, University of Toronto, Department of Computer Science. Retrieved on July 27, 2007 from http://www.cs.utoronto.ca/~jenhork/MScThesis/Thesis.pdf.

Horkoff, J. (2004). T Montreux Jazz Festival i* Exercise presentation. Based on Osterwalder Ph.D. Thesis.

Horkoff, J., Yu, E., & Liu, L. (2006). Analyzing Trust in Technology Strategies presentation. Privacy, Security, Trust 2006. Markham, Ontario, Canada. October 31, 2006October 2006.

Horkoff, J. and Yu, E. (2005). Using the i* Evaluation Procedure for Model Analysis and Quality Improvement presentation. Second International Workshop on i* / Tropos. University College London, London UK.

Horkoff, J., Yu, E., & Liu, L. (2004). Trusted Computing: An i* Case Study presentation. Department of Computer Science. University of Toronto.

[NFRbook] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. Non-Functional Requirements in Software Engineering (Monograph) Kluwer Academic Publishers, 2000. 472 pp.

[PST06] J. Horkoff, E. Yu, L. Liu. (2006). Analyzing Trust in Technology Strategies. . Int. Conf. on Privacy, Security, and Trust (PST'06), Toronto, Canada, Oct 30— Nov 1, 2006.

[SREIS02-Sec] L. Liu, E. Yu, J. Mylopoulos. Analyzing Security Requirements as Relationships Among Strategic Actors. 2nd Symposium on Requirements Engineering for Information Security (SREIS'02). Raleigh, North Carolina, October 16, 2002

Yu. E. (2006). Computer Science Department. University of Toronto. Prof. Eric Yu Website. Retrieved on July 28, 2007 from http://www.cs.toronto.edu/%7Eeric/#Research%20Interests.

[Yu01] Yu, Eric. Presentation: Strategic Actor Relationships Modelling with i*, December 13-14, 2001, IRST, Trento, Italy.

[Yu95] E. Yu. Modelling Strategic Relationships for Process Reengineering Ph.D. Thesis. Dept. of Computer Science, University of Toronto. 1995.

The evolution of i* syntax, deviating from the original description in [Yu95] has occurred over the course

of many papers. Such work can be found in the i* Roadmap.

Created by: system last modification: Wednesday 12 of March, 2008 [15:44:29] by samer

The original document is available at http://http://istar.rwth-aachen.de//tiki-index.php?page=iStarGuide