UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO
EM COMPUTAÇÃO APLICADA

Júnior Machado Martins

RequirementX: uma ferramenta para suporte à Gerência de Requisitos em Extreme Programming baseada em Mapas Conceituais

São Leopoldo
2007

Júnior Machado Martins

RequirementX: uma ferramenta para suporte à Gerência de Requisitos em Extreme Programming baseada em Mapas Conceituais

Monografia apresentada à Universidade do Vale do Rio dos Sinos como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Orientador: Prof. Dr. Sérgio Crespo C. S. Pinto

São Leopoldo

2007

Ficha catalográfica elaborada pela Biblioteca da

Universidade do Vale do Rio dos Sinos

M149r Martins, Júnior Machado

RequirementX: uma ferramenta pra suporte à gerência de requisitos em *extreme programming* baseada em mapas conceituais / por Júnior Machado Martins. – 2007.
113p. : il. ; 30cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2007.

"Orientação: Prof. Dr. Sérgio Crespo C. S. Pinto, Ciências Exatas e Tecnológicas".

1. Engenharia de *software*. 2. *Extreme programming.* 3. XP. 4. Mapa conceitual. 5. Ontologia. I. Título.

CDU 004.41

Catalogação na Publicação:

Bibliotecária Eliete Mari Doncato Brasil - CRB 10/1184

Mensagem

"Trabalhe com os bons, trabalhe com os campeões, um campeão vai te ensinar a ser um campeão".

Roberto Shinyashiki

Agradecimentos ...

Agradeço a Deus, por ter me dado saúde, paz, proteção, força, vontade, convicção e tudo mais que precisei para concluir esta etapa.

Agradeço aos meus pais, Alderi e Orfelina, e a meu irmão, Fabiano, pelo carinho, apoio e confiança durante toda a minha vida, e por serem minha base.

Agradeço em especial, novamente, ao meu pai, meu primeiro orientador, que me incentivou a estudar e sempre me deu exemplo de caráter, honestidade, trabalho, força, fé, amor e determinação.

Agradeço aos meus avós, pelo amor e carinho dado durante toda a minha vida, e por sempre torcerem pelos meus projetos.

Agradeço aos meus padrinhos, pelo carinho, incentivo e conselhos dados durante toda a minha vida.

Agradeço ao meu orientador e amigo, Prof. Dr. Sérgio Crespo, pela grandiosa orientação neste trabalho, pelas oportunidades e conselhos que me ofereceu durante o curso e por tudo que me ensinou.

Agradeço aos meus amigos e colegas, em especial Letícia Rafaela, Marcelo Scopel, Ricardo Kratz, Genessa Robinson, Elisângela, Patrícia, Galina, Tiago, André, Rogério, Andresa e a todos os demais pela amizade, companheirismo e contribuição no desenvolvimento deste trabalho.

Agradeço a todo o pessoal do Instituto IHMC na Flórida, pela grandiosa contribuição em minha pesquisa e pela hospitalidade e carinho com que me receberam.

Agradeço também à Tia Leda, minha melhor amiga, Tia Eva, Tia Sueli e a todos os demais familiares, pelo grande carinho que sempre me deram durante esta caminhada.

Agradeço aos nobres amigos Dr. Oly, Mari e sua família, pelo carinho e contribuição em meus desafios.

Agradeço também a todos meus amigos, que são muitos, por sempre estarem presentes em minha vida.

Resumo

Uma das tarefas críticas na confecção de sistemas de *software* é a elicitação de requisitos, a qual configura uma ação de descoberta de conhecimento. Assim, muitas técnicas são empregadas na tentativa de minimizar conflitos de idéias, conceitos mal formados, interpretações redundantes e omissão de dados; sendo que, para tanto, o uso de cenários, entrevistas, cartões, *viewpoints* e diagramas de *Use Case* são utilizados como ferramentas para diminuir a distância entre o técnico e o usuário na definição dos requisitos. Além disso, os Mapas Conceituais têm sido empregados com muita eficiência em tarefas de captura de conhecimento, portanto, este trabalho utiliza esse conceito como forma de organizar, identificar, aprimorar conceitos e definições dos requisitos de um *software* de forma cooperativa, formatado em *User Story* da metodologia *Extreme Programming* (XP). Com esse objetivo, o processo é apoiado por uma ferramenta baseada na *web*, que automatiza a geração, organização e acompanhamento da captura dos requisitos gerados a partir de Mapas Conceituais.

Palavras-chave: *Extreme Programming* (XP), *User Story*, Mapas Conceituais, *CmapTools*, *Web* Semântica, RDF, Ontologia, XML, OWL, UML, e UML-MC.

Abstract

One of the hardest tasks of building a software system is requirements elicitation, which triggers a knowledge discovery action. Thus, many techniques are used with the intention to minimize idea conflicts, misformed concepts, erroneous interpretations and missing data; In order to achieve this goal, scenarios interviews, User Stories, viewpoints and Use Case diagrams are techniques to reduce the distance between the researcher and the user on requirement elicitation. Concept maps have been used as efficient way to represent knowledge. This research uses concept maps to deal with the organization, identification and improvement of concepts and software requirements definitions in a cooperative way, making use of the User Story format introduced by the Extreme Programming (XP) methodology. The proposed process is supported by a web-based tool, which automates the generation, organization and management of the requirements capture generated in the Concept Maps format.

Keywords: Extreme Programming (XP), User Story, Conceptual Maps, CmapTools, Semantic Web, RDF, Ontology, XML, OWL, UML, and UML-MC.

Índice de tabelas

Índice de figuras

Lista de abreviaturas e siglas

CMAP – Mapas Conceituais

CVS – Sistema de Versões Concorrentes

DAML – acrônimo para a linguagem *DARPA Agent Markup Language*

EDT – Estrutura de Desmembramento de Trabalho

GUI – Interface de Uso Gráfico

IBID – Significa "na mesma obra"

OIL – acrônimo para a linguagem *Ontology Inference Layer*

OWL - acrônimo para a linguagem *Ontology Web Language*

PERT – Técnica de Avaliação e Revisão de Programas

RDF – acrônimo para a linguagem *Resource Description Framework*

UML – Linguagem de Modelagem Unificada

URI – *Uniform Resource Identifier*

URL – Localizador Uniforme de Recursos

WEB – Rede de alcance mundial ou "WWW" (*World Wide Web)*

W3C – *World Wide Web Consortium*, é um consórcio de empresas de tecnologia (www.w3c.org).

XML – *Extensible Markup Language,* é uma linguagem de marcação recomendada pelo consórcio W3C.

XP – Programação Extrema

Índice

1 Introdução

A Engenharia de Requisitos é uma subárea da Engenharia de *Software*, e procura definir um formalismo para a representação dos requisitos de *software*; sendo que sobre isso Brooks [BRO 1978] comenta que "a única parte mais difícil de construção de um sistema de *software* é decidir precisamente o que construir. Nenhuma outra parte do trabalho conceitual é tão difícil como estabelecer os requisitos técnicos detalhadamente. Nenhuma parte é mais difícil para retificar depois". Assim, através da história da indústria desse ramo, temos nos deparado constantemente com esta verdade, e, por isso, definir e aplicar requisitos bons e completos é um trabalho difícil, e o sucesso nessas tentativas não é constante no trabalho de muitos desenvolvedores, embora tem havido progresso. Por exemplo, em muitos campos, pesquisadores propõem novos métodos para tratar o problema e, eventualmente, alguns deles são adotados, mas a falta de completa adoção é conhecida como a distância entre a pesquisa e a prática [LEI 2005]. Um exemplo é a *Extreme Programming* (XP) que possui uma forma peculiar de tratar a documentação e organização dos requisitos, chamada *User Stories* [XIS 2005],[BEC 1999] e [BEC 2000], através de um processo que deve ser ágil e eficaz. Porém, o que se percebe é que o processo não acompanha a metodologia como um todo, deixando enfraquecido esse importante ponto [XIS 2005].

Vamos descrever a seguir o que motivou o desenvolvimento desta pesquisa, uma descrição do problema, o objetivo e a estrutura do trabalho.

1.1 Motivação e contexto do trabalho

Com o advento da *Internet*, o processo de desenvolvimento de *software* mudou drasticamente, partindo de um modelo mais estático para um modelo mais flexível e dinâmico, em que as mudanças nos requisitos do sistema mudam com maior freqüência. Por essa razão, metodologias ágeis, tais como *Extreme Programming* (XP), a qual, segundo Kent Back, seu mentor, "é uma metodologia ágil para equipes pequenas e médias desenvolvendo *software* com requisitos vagos e em constante mudança", têm causado grande impacto no cenário mundial de desenvolvedores de *software* [BEC 1999].

Portanto, a metodologia carece de ferramentas de documentação do processo de descoberta dos requisitos do sistema que nortearão todo o processo de desenvolvimento do produto final. E isso, conseqüentemente, deixa-nos em posição de repensar as ferramentas ou artefatos e implementar uma nova proposta de trabalhar a descoberta de requisitos de forma simples, cooperativa, que permita constante atualização, e que reforce a comunicação do grupo, sendo que assim este trabalho enfatiza tais aspectos.

1.2 Problema

O problema investigado aborda o modo como os requisitos de um produto de *software* podem ser representados e documentados em metodologias ágeis, em especial na *Extreme Programming*, usando Mapas Conceituais.

1.3 Objetivo

O principal objetivo deste projeto é a aplicação da teoria de Mapas Conceituais no processo de elicitação de requisitos em *Extreme Programming*, de forma a obter uma formalização do processo e a permitir que os requisitos dos atores envolvidos possam ser rapidamente modelados e atualizados de forma cooperativa, por meio de uma ferramenta protótipo chamada RequirementX.

1.4 Organização do trabalho

Este volume está dividido em seis capítulos, sendo que o primeiro é a corrente introdução, e que os demais são descritos a seguir:

- Capítulo 2: Revisão bibliográfica - descreve as tecnologias utilizadas no desenvolvimento do trabalho, sendo que o texto especifica o objetivo do problema e discute sobre algumas técnicas de elicitação de requisitos relevantes para este trabalho, a *Extreme Programming*, os Mapas Conceituais, o *CmapTools*, a gerência de projetos, a EDT (Estrutura de Desmembramento de Trabalho), a *Web* Semântica e também a UML-MC;
- Capítulo 3: RequirementX - descreve a solução a ser desenvolvida e aplicada no estudo de caso proposto no próximo capítulo;
- Capítulo 4: Estudo de caso - mostra a aplicação da solução proposta com a utilização de um cenário baseado em uma regulamentação do governo americano que uma empresa deve seguir;
- Capítulo 5: Trabalhos relacionados - discute algumas soluções cujos objetivos são semelhantes. Além disso, ao final do capítulo há uma comparação dessas abordagens com a RequirementX;
- Capítulo 6: Conclusão - apresenta algumas considerações quanto ao desenvolvimento do trabalho, indicando pontos de melhoria e também a continuidade do seu desenvolvimento.

2 Revisão bibliográfica

Este capítulo descreverá as pesquisas realizadas e os estudos envolvendo conceitos e métodos destinados à conclusão do propósito geral deste trabalho. Então, com esse objetivo, será apresentada aqui uma revisão bibliográfica sobre o objetivo do problema, técnicas de elicitação de requisitos, Engenharia de Requisitos, *Extreme Programming* (XP), Mapas Conceituais, *CmapTools*, Gerência de Projetos, *Web Semantic* e UML-MC.

2.1 Especificando o objetivo do problema: o processo de elicitação de requisitos

A elicitação de requisitos é um processo que envolve atividades para descobrir requisitos de um determinado sistema, sendo que para isso, os desenvolvedores do sistema trabalham junto com os *stakeholders* - termo para fazer menção a alguém que pode ter intervenção direta ou indireta sobre os sistemas – para que atenda às necessidades identificadas [SUZ 1999].

Além do mais, esse processo apresenta algumas dificuldades que são caracterizadas como [SUZ 1999] [HER 2002]:

- Os *stakeholders* expressam os requisitos do sistema através de seus próprios termos, o que às vezes dificulta a tarefa dos engenheiros de requisitos;
- Os requisitos especificados apresentam diferentes caminhos para serem atingidos, e cabe aos engenheiros verificar as possibilidades, identificando as semelhanças e os conflitos;
- Os fatores políticos são influências observáveis principalmente em cargos de gerência;
- Os requisitos podem ser alterados em vista do mundo dinâmico dos negócios.

Já para Leffingwell [LEF 2000], as dificuldades mencionadas podem ser classificadas como síndromes, dentre as quais se destacam três estilos, os quais são descritos a seguir:

1) Síndrome do "*Yes, But*": parte do desenvolvimento das aplicações, e acontece quando o usuário inicialmente concorda com as colocações do engenheiro de *software*, afirmando que a solução apresentada resolverá todos os seus problemas. No entanto, o cliente retoma sua posição e faz questionamentos sobre tudo o que havia concordado anteriormente.

2) Síndrome "*Undiscovered Ruins*": identifica as ruínas escondidas, ou seja, no momento em que se efetua um processo de elicitação, encontram-se muitos requisitos. Mas, como não existe um número limite de requisitos para um determinado problema, o desenvolvedor provavelmente nunca irá encontrar todos os possíveis requisitos de um sistema, mas sim, uma quantidade suficiente para a sua implementação.

3) Síndrome "*User and the Developer*": comprova-se com a comunicação entre os usuários e desenvolvedores, cujos vocabulários são distintos, uma vez que eles utilizam

linguagens diferentes, sentem motivações e almejam objetivos muitas vezes diferenciados.

Portanto, o processo de elicitação está relacionado com a análise e com a negociação de requisitos, cujo alvo é estabelecer requisitos completos e consistentes através da reestruturação dos conflitos descobertos. Segundo Pressman [PRE 1995], a comunicação nessa fase é elevada e a proporção de interpretações errôneas e informações falsas são observadas.

Sendo assim, para um melhor detalhamento desse processo são identificadas quatro dimensões que compõem o processo de elicitação, que são: a compreensão do domínio de aplicação, a compreensão do problema, a compreensão do negócio e a compreensão das necessidades e restrições dos *stakeholders* [SUZ 1999].

Ainda é importante salientarmos que a aceitação do sistema depende do atendimento de tais necessidades, ou seja, o processo em questão deve sempre buscar novas informações, já que a economia e o mundo dos negócios estão em constantes transformações. Dessa forma, utilizam-se técnicas para analisar e desenvolver possíveis soluções para as dificuldades encontradas, as quais são, geralmente, aplicadas no desenvolvimento de todos os tipos de sistema.

2.2 Técnicas de elicitação de requisitos

Entre as técnicas de elicitação aplicadas destacam-se as seguintes: entrevista, *workshops*, *brainstorming*, cenários [BRE 1999], *storyboards*, etnografia, *role playing*, protótipos e *document archeology* [PRE 1995]. Portanto, como pode ser observado na literatura, existem problemas referentes ao tempo de execução do trabalho, à falta de clareza, à não existência de cooperação na construção dos diversos artefatos de requisitos, e a uma fraca intercomunicação do grupo de pessoas envolvidas, que conseqüentemente são responsáveis pelo sistema [BEU 2003] e [BEB 2003].

As técnicas de elicitação de requisitos têm como objetivo gerar e consolidar idéias dos *stakeholders* do sistema, sendo que, inicialmente, elas representarão os requisitos de negócio e dos usuários. Posteriormente, entretanto, a equipe de desenvolvimento poderá refinar essas idéias, transformando-as em requisitos de sistemas a serem desenvolvidos.

Sendo assim, neste capítulo vamos apresentar as principais características de algumas técnicas de elicitação de requisitos, e, por fim, apresentar as vantagens de nossa proposta em relação a essas outras técnicas.

2.2.1 Entrevista

Entrevistas são as técnicas de elicitação mais utilizadas, e são praticamente inevitáveis em qualquer desenvolvimento, já que é uma das formas de comunicação mais natural entre pessoas. No entanto, elas não são tão simples como inicialmente podem parecer, uma vez que entrevistar não significa somente fazer perguntas. Assim, ela é uma técnica estruturada que pode ser aprendida, sendo que os desenvolvedores podem conquistar a proficiência com treinamento e prática, pois as entrevistas são usadas para obter requisitos sobre um domínio através de perguntas feitas aos usuários especialistas desse

domínio por um ou mais entrevistadores. Além disso, elas permitem também que os desenvolvedores entendam os processos atuais da organização, percebam o que está faltando no sistema existente, e quais são as expectativas dos usuários do novo sistema. Finalmente, é importante citar que as entrevistas podem ser divididas em três tipos: as não estruturadas, as semi-estruturadas e as estruturadas [CAR 2001].

2.2.2 Workshops

Os *workshops* são encontros em que os principais *stakeholders* do projeto se reúnem por um período curto de tempo, sendo que durante esse tempo, eles ficam totalmente focados na atividade em questão. Além disso, os *workshops* geralmente são realizados fora do ambiente de trabalho para evitar que os *stakeholders* sejam interrompidos com freqüência, e são geralmente conduzidos por um membro da equipe de desenvolvimento ou por um facilitador externo, que tenham experiência nesse tipo de reunião. Sendo assim, o *workshop* tem como objetivo principal fomentar a discussão organizada entre os *stakeholders*, e suas principais vantagens são:

- Eles ajudam na criação de uma equipe com um objetivo claro de fazer com que o projeto tenha sucesso;
- Todos os *stakeholders* têm direito de expor sua opinião, sem que ninguém prevaleça sobre os demais;
- O *workshop* ajuda a formar um compromisso entre a equipe e os *stakeholders* sobre o quê o sistema deve fazer;
- Ele pode proporcionar a exposição e a resolução de problemas políticos que possam interferir no sucesso do projeto.

Ainda é importante dizer que a preparação do *workshop* é essencial, pois se os *stakeholders* não estiverem confiantes nos resultados que podem ser gerados, o *workshop* não terá efeito positivo. Assim, a preparação consiste em identificar os *stakeholders* que podem contribuir mais fortemente para o projeto. Após identificá-los, o conceito do *workshop* deve ser "vendido" para eles. Além disso, mesmo que o material não seja lido, ele ajuda os *stakeholders* a verem o *workshop* como uma reunião séria e planejada [LEF 2000].

2.2.3 Brainstorming

O *brainstorming* é uma das diversas técnicas de reuniões de grupo, e provavelmente a mais antiga e mais conhecida, por ser uma técnica básica para geração de idéias; e cujo princípio básico é reunir um conjunto de especialistas (de sistemas e de negócios) para que cada um possa inspirar o outro na criação de idéias que contribuam para resolver o problema em uma ou várias reuniões, sendo que as idéias sugeridas e exploradas nesses encontros não devem ser criticadas ou julgadas. Sendo assim, a técnica pode ser aplicada no início da fase do desenvolvimento quando pouco do projeto é conhecido e são necessárias idéias novas, uma vez que ela é usada exatamente para gerar novas

idéias, deixando a mente livre para aceitar toda a idéia que for sugerida e permitir liberdade para a criatividade. Portanto, o resultado de uma sessão de *brainstorming* bem sucedida é um conjunto de boas idéias e a sensação de que todos participaram da solução do problema, sendo ela uma técnica particularmente efetiva para ser aplicada à concepção de um sistema ou na exploração e entendimento do potencial de mercado para o produto em desenvolvimento [BAT 2005].

2.2.4 Storyboards

Essa técnica envolve a descrição através de quadros com imagens que ilustram as situações do domínio, sendo que cada quadro deve apresentar a cena que descreve a situação, os atores e as ações que cada um deve desempenhar; e que abaixo de cada quadro devem estar descritas ações que os atores desempenham. Dessa forma, os *storyboards* são bastante utilizados em cinema como forma de comunicação entre roteiristas, diretores, atores e técnicos, além de ser uma técnica bastante relacionada com cenários, por permitir a descrição de situações de uso [LEI 2000].

2.2.5 Protótipos

Protótipos são implementações parciais de um sistema de *software*, que são construídas para ajudar os desenvolvedores, usuários e *stakeholders* a entender melhor os requisitos do sistema, e podem ser vistos como um tipo de *storyboarding*, só que geralmente são mais sofisticados, pois envolvem interação com o usuário [LEF 2000] [PRE 1995] .

Além disso, os protótipos podem ser classificados como:

- Descartável ou evolutivo: protótipos descartáveis são jogados fora depois de apresentados ao usuário; e protótipos evolutivos são refinados para se transformarem, progressivamente, no sistema desejado.
- Vertical ou horizontal: protótipos verticais têm como objetivo identificar os detalhes de uma determinada operação no sistema; e protótipos horizontais visam a apresentar um grande número de operações, analisando sua integração.

2.2.6 Role playing

*Role playin*g é uma técnica em que o desenvolvedor trabalha algum tempo nas atividades realizadas pelo usuário, permitindo que ele veja o mundo pelos olhos dos seus usuários. Em outras palavras, durante a sessão de *role playing*, o desenvolvedor pode viver os problemas e incertezas que o usuário enfrenta no dia-a-dia, o que ajuda no desenvolvimento de um projeto.

Porém, nem sempre é possível atuar como o usuário, pois muitas vezes o seu papel é pequeno em um sistema, sendo que os algoritmos são os maiores problemas. Além disso, para exemplificar essa situação, pensemos nas seguintes questões: quantos de nós

seríamos candidatos a fazer o papel de paciente em um sistema de controle cirúrgico? Ou o papel de operador de um sistema de controle de usina nuclear? Ou o papel de piloto de um 767?

Portanto, quando não for possível substituir o usuário em um posto de trabalho, outras técnicas podem simular o *role playing,* como por exemplo os cenários, que são simulações como teatro ou RPG [LEF 2000].

2.2.7 Cenários

Os métodos baseados em cenários consistem em uma coleção de narrativas de situações no domínio que favorecem o levantamento de informações, a identificação de problemas e a antecipação das soluções. Por essa razão, os cenários são maneiras excelentes de representar, para clientes e usuários, os problemas atuais e as possibilidades que podem surgir.

Os cenários têm como foco as atividades que as pessoas realizam nas organizações, e, então, possibilitam uma perspectiva mais ampla dos problemas atuais onde o sistema será inserido, explicando porque ele é necessário; além de proporcionarem também um desenvolvimento orientado a tarefas, o que possibilita maior usabilidade do sistema.

No entanto, não é objetivo dos cenários oferecerem uma descrição precisa, mas provocar a discussão e estimular novos questionamentos, uma vez que eles permitem também documentar o levantamento de informações a respeito dos problemas atuais, dos possíveis eventos, das oportunidades de ações, e dos riscos.

Por sua simplicidade, os cenários são um meio de representação de fácil compreensão para os clientes e usuários envolvidos (muitas vezes de formação bastante heterogênea), os quais podem avaliar, criticar e fazer sugestões, proporcionando a reorganização de componentes e tarefas do domínio. Além do mais, os cenários oferecem um "meio-intermediário" entre a realidade e os modelos que serão especificados, possibilitando que clientes, usuários e desenvolvedores participem do levantamento das informações e especificação dos requisitos. Em resumo, os cenários permitem compreensão dos problemas atuais pelos desenvolvedores e antecipação de situações futuras pelos clientes e desenvolvedores [BRE 1999] [LEI 2000].

2.2.8 Etnografia

Nessa abordagem, as pessoas que irão efetuar a aquisição do requisito integram-se ao Universo de Informações de forma a terem um conhecimento o mais amplo possível sobre o sistema de informação vigente na organização, sendo que essa integração deve acontecer de tal forma que quem irá obter as informações passe a ser visto como um usuário, isto é, podendo ter as mesmas perspectivas que teria um usuário. Além disso, a principal vantagem dessa técnica é a possibilidade de uma visão de dentro para fora mais completa e perfeitamente ajustada ao contexto, evitando, assim, o problema do conhecimento tácito, pois as pessoas não precisam descrever o que fazem e sim demonstrar o quê e como fazem. Já no que diz respeito às desvantagens, as principais

referem-se ao tempo gasto e também à pouca sistematização do seu processo [BOR 2005].

2.3 Engenharia de Requisitos

O termo "*Requirement Engineering*" é traduzido para o português como "Engenharia de Requisitos", e significa a aplicação de princípios científicos às construções. Por sua vez, entende-se por requisito a condição que se precisa para conseguir certo fim, como a exigência legal necessária para certos efeitos [ZAN 1998]. A seguir serão apresentados outros conceitos no contexto de Engenharia de Requisitos.

Para Sommerville [KOT 1998] [SOM 1997], Engenharia de Requisitos é um termo relativamente novo, que foi inventado para cobrir todas as atividades envolvidas em descobrimento, documentação e manutenção do conjunto de requisitos para um sistema baseado em computador. O uso do termo engenharia implica em técnicas sistemáticas e repetíveis a serem usadas para assegurar que os requisitos do sistema sejam completos, consistentes, e relevantes. Por sua vez, o termo Engenharia de Requisitos vem da antecedente Engenharia de Sistemas, sendo que ambas são correspondentes à fase de análise de sistemas.

Para Macaulay [MAC 1996], Engenharia de Requisitos pode ser definida como o processo sistemático de desenvolvimento de requisitos através de um processo iterativo de análise do problema, documentação de observações resultantes em uma variedade de formatos de representação e checagem de precisão do entendimento obtido; e pode ser considerada como o processo através do qual o documento de requisitos é preenchido.

2.3.1 Processo de Engenharia de Requisitos

O processo de Engenharia de Requisitos [MAC 1996] apresenta dois tipos básicos de atividades, as quais são descritas a seguir:

- Análise do problema – durante essa atividade, os desenvolvedores usam seu tempo em *brainstorming* e entrevistas com pessoas que têm mais conhecimento acerca do problema, e identificam as possíveis restrições sobre a solução do problema. Além disso, ela é caracterizada pela incerteza, e pela expansão da informação e do conhecimento.
- Descrição do produto – esse é o tempo de escrever, tomar algumas decisões difíceis e preparar um documento que descreva o comportamento externo esperado do produto. Além disso, essa atividade é caracterizada pela organização de idéias, resolução de conflitos de pontos de vista e intenções dos clientes, da variedade de cenários e eliminações de inconsistências e de ambigüidades, sendo que não existe um modelo único de processo, mas muitos.

Também é importante dizer que um processo de Engenharia de Requisitos [SOM 1997] é um conjunto estruturado de atividades para extrair, validar e manter um documento de requisitos, sendo que uma descrição completa do processo poderá incluir quais

atividades são realizadas, assim como também sua estruturação ou particionamento, quem é responsável pela atividade, quais são as entradas e saídas de/para atividade, e ainda quais são as ferramentas usadas para suportar a Engenharia de Requisitos.

2.3.2 Domínio da aplicação ou ambiente

O primeiro passo em análise de problemas é estruturar e analisar o domínio da aplicação (ou ambiente), sendo que o princípio de relevância do domínio declara que "tudo o que é relevante para os requisitos, deve aparecer em alguma parte do domínio da aplicação". Assim, existe uma importante conseqüência desse princípio, uma vez que o domínio da aplicação não é limitado a partes do mundo que estão diretamente ligados ao sistema de *software*, mas é mais abrangente, e refere-se ao negócio [JAC 1995]. Então, é possível afirmarmos que o domínio da aplicação é onde os requisitos particulares dos clientes são encontrados, assim, se o domínio da aplicação não for identificado corretamente, não se está apto para focalizar os requisitos dos clientes.

Em cada atividade de desenvolvimento de *software*, o contexto do problema tem pelo menos dois domínios distintos: o da aplicação (o ambiente ou o mundo real) e o sistema de *software*. Sendo assim, o domínio da aplicação é onde os requisitos do cliente existem. Portanto, pode-se pensar do domínio da aplicação como o que é dado, e do domínio do *software* como o que será construído [JAC 1995].

Além disso, é importante que o conceito de requisitos, programas e especificações sejam estabelecidos. Sendo assim, requisitos são exclusivamente fenômenos observados no ambiente, isto é, os fenômenos externos ao *software*. Por sua vez, programas são, exclusivamente, todos os fenômenos de *software*, isto é, os fenômenos internos à máquina. E, finalmente, especificações são, exclusivamente, todos os fenômenos compartilhados (estados e eventos), que formam a interface entre o ambiente e o *software*, conforme apresentado na Figura 1.

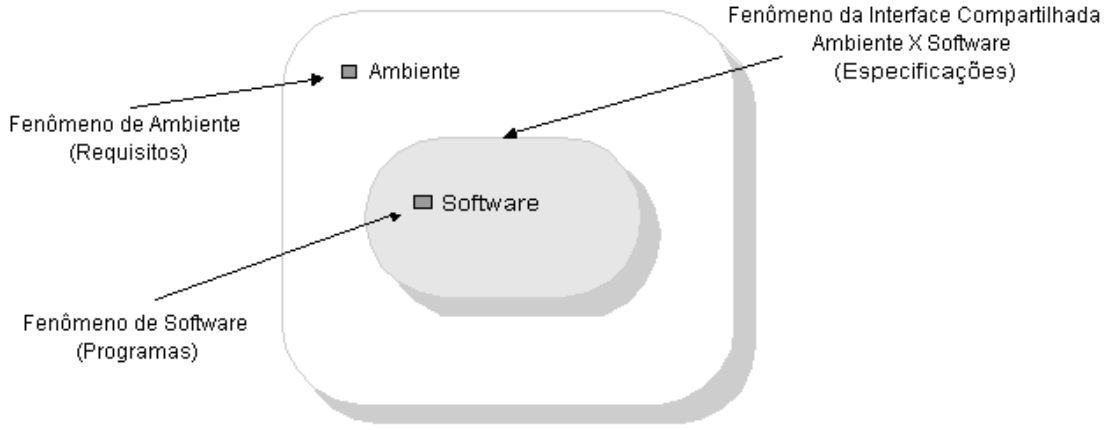


Figura 1 - Domínio da aplicação (ambiente) X *Software* [JAC 1995]

2.3.3 Requisitos

Requisito pode ser definido simplesmente como “algo que um cliente necessita”. Entretanto, do ponto de vista de um desenvolvedor, requisito pode também ser definido como “algo que necessita ser projetado” [MAC 1996].

No entanto, existem inúmeras definições do termo “requisitos” [MAC 1996]. A seguir veremos três dessas definições:

a. Uma condição ou capacidade necessária para um cliente desenvolver um problema ou realizar um objetivo;

b. Uma condição ou capacidade que deve ser encontrada ou possuída por um sistema ou componente de sistema, para satisfazer um contrato, um padrão, uma especificação ou um outro documento imposto formalmente;

c. Uma representação documentada de uma condição ou capacidade, como em “a” ou “b” acima.

Além disso, requisitos são definidos como: as descrições de como o sistema poderá comportar-se, as informações do domínio da aplicação, as condições sobre operação de sistema ou, ainda, como as especificações de uma propriedade ou atributo de sistema. Os requisitos são definidos durante os estágios iniciais do desenvolvimento de um sistema como uma especificação do que poderá ser implementado, e contêm, invariavelmente, uma mistura de informação do problema, das declarações de comportamento e propriedades do sistema, e das condições de projeto e construção. Porém, essa abordagem de definição é exclusiva para sistema de *software*, notadamente voltada para o sistema como solução para o problema [KOT 1998] [SOM 1997].

Uma outra definição diz ainda que requisitos são fenômenos do domínio da aplicação. São exclusivamente todos os fenômenos do ambiente. É uma propriedade do domínio da aplicação, que o sistema de *software* deve executar, sendo que, para descrevê-los exatamente, descrevemos os relacionamentos acerca dos fenômenos do contexto do problema. Podemos concluir que muitos projetos falham porque seus requisitos estão inadequadamente explorados e descritos [JAC 1995] [ZAV 1997].

2.3.4 Especificações

Os requisitos [SOM 1997] caracterizam-se sob duas formas fundamentais:

- Funcionais – são os fenômenos ambientais referentes ao negócio da aplicação que se queira conhecer e estudar. Assim, eles são descritos do ponto de vista do cliente, e normalmente são expressos em linguagem natural, num diagrama informal ou usando alguma notação que é apropriada para o entendimento do problema.

- Não-funcionais – são as especificações técnicas de como melhor adequar a solução do problema para responder ao cliente, e podem ser expressas como um modelo abstrato do sistema. Além disso, esse modelo pode ser um modelo matemático ou baseado em notações gráficas, tais como diagrama de fluxo de dados, hierarquia de classes de objetos, e de preferência em linguagens de representação formal, que podem gerar código fonte e objeto diretamente.

Entretanto, a terminologia de desenvolvimento de *software* não é padronizada e o conceito de especificação é um dos elementos que fazem parte do universo do contexto da Engenharia de *Software*. Dentre eles, a que mais se aproxima e restringe a forma de pensar diz o seguinte: "Em geral, pode-se ver uma especificação como uma declaração de acordo entre um produtor e um consumidor de serviço ou entre um implantador e um usuário" [JAC 1995].

2.4 *Extreme Programming (XP)*

A XP é uma metodologia ágil para equipes pequenas e médias desenvolvendo *software* com requisitos vagos e em constante mudança, que prima pela satisfação do cliente e pela qualidade do *software* final, causando, com isso, grande comoção num mercado de desenvolvimento de *software* cansado do alto custo das metodologias pesadas tradicionais e do notório fracasso da maioria de seus projetos [BEC 1999].

Dessa forma, os seguidores desse paradigma de desenvolvimento adotam valores, princípios e práticas, sendo que os valores são quatro lemas que correspondem a quatro dimensões nas quais os projetos devem ser melhorados, e os quais são: comunicação, simplicidade, *feedback*, e coragem. Sendo assim, esses valores dão os critérios para a solução de sucesso, mas, no entanto, são muito vagos para nos ajudar a decidir quais práticas usar. E, por essa razão, devem ser transformados em princípios concretos que possam ser usados.

Por sua vez, esses princípios vão ajudar a escolher entre as alternativas, sendo que será dada preferência àquelas que melhor atenderem aos princípios, pois cada princípio incorpora os valores, e um valor pode ser vago. Além disso, o que é simples para uma pessoa pode ser complexo para outra, e, ainda, um princípio pode ser mais concreto, por exemplo, ou um *feedback* pode ser rápido ou não. Então, os princípios fundamentais são os seguintes:

- *Feedback* rápido;
- Simplicidade presumida;
- Mudanças incrementais;
- Aceitação das mudanças;
- Alta qualidade.

Ainda é importante salientar que a XP possui um conjunto de doze práticas, sendo que elas se apoiam umas às outras, ou seja, o ponto fraco de uma é compensado pelos pontos fortes das outras. As práticas da XP são [AST 2002] [SCH 2003] [OSH 2005]:

1. Jogo de planejamento;
2. *Releases* pequenos;
3. Metáforas;
4. Projeto simples;
5. Testes de aceitação;

6. Refatoração;
7. Programação em pares;
8. Propriedade coletiva;
9. Integração contínua;
10. Semana de trabalho de quarenta horas;
11. Cliente no local;
12. Padrões de Código [BEC 1999].

O ciclo de desenvolvimento dentro do paradigma XP é representado de forma simplificada através de quatro fases, as quais são demonstradas na Figura 2 abaixo:

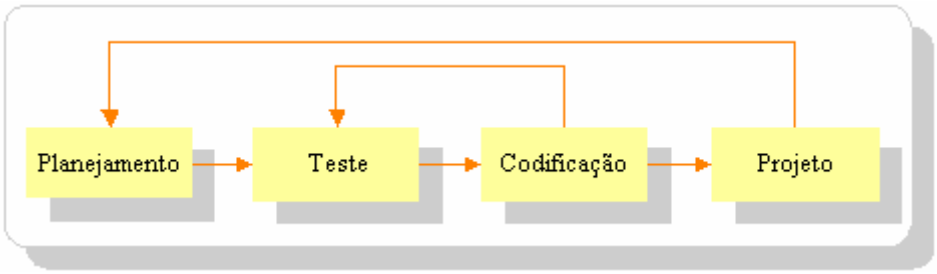


Figura 2 - Ciclo de vida no modelo XP [BEC 1999]

Finalmente, devemos citar que, durante a fase de planejamento, diversos fatores que podem afetar o desenvolvimento são estimados, mas, como a XP não define uma especificação formal e completa do modelo, esses fatores são resumidos aos itens citados abaixo:

- *User Story*;
- Lançamento do produto;
- Métricas;
- Planejamento das iterações de desenvolvimento;
- Reuniões de Pé.

2.4.1 O cartão de visão

Um cartão de visão [AST 2002] é uma declaração com vinte e cinco palavras ou menos sobre a finalidade da criação ou ampliação do sistema, sendo que os clientes são autores dessa declaração de missão. Portanto, é possível que o cliente escreva mais, entretanto, é mais difícil, embora seja melhor, escrever uma descrição concisa de um sistema do que escrever um texto muito longo.

Assim, quando escreve um cartão de visão, um cliente tem de pensar sobre os motivos pelos quais precisa ter o sistema, uma vez que o cartão também define a tônica do projeto.

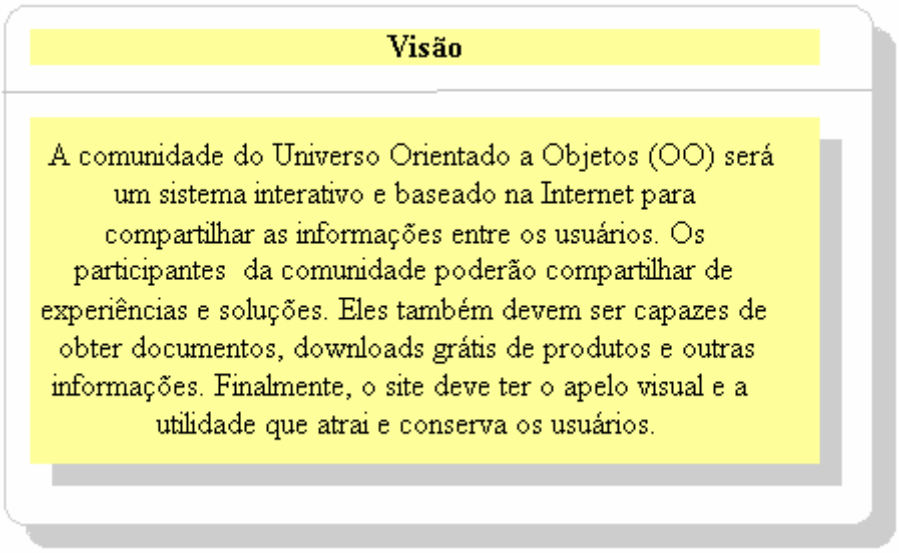


Figura 3 - Exemplo de um cartão de visão [AST 2002]

Portanto, o cartão de visão descreve a finalidade do sistema, sendo que o uso das metáforas pode ampliar essa visão para desenvolver uma compreensão conceitual do sistema entre os clientes e os desenvolvedores. Então, a criação de um consenso sobre essa visão inicial é crítica para a próxima etapa, na qual são escritas as *User Stories*, as quais, por sua vez, refinam ainda mais essa visão em unidades que podem ser estimadas e testadas.

2.4.2 *User Stories*

A XP utiliza, dentre as suas práticas, a *User Story* para realizar a documentação dos requisitos e observações provenientes das reuniões com os usuários e pessoal técnico.

Isso porque elas são usadas para criar estimativas de tempo para a reunião de planejamento de *release*, e também são usadas no lugar de um documento enorme descrevendo os requisitos. Além disso, são escritas pelos clientes como atividades que o sistema precisa fazer para eles, e têm o mesmo propósito que as Casos de Uso da UML, embora não seja o mesmo, pois elas não estão limitadas a descrever uma interface com o usuário. Ainda, o formato delas é cerca de três sentenças de texto escritas pelo cliente, na terminologia do cliente, sem uma sintaxe específica; e também conduzem à criação dos testes de aceitação, sendo que um ou mais testes de aceitação devem ser criados para verificar se elas foram corretamente implementadas [BEC 1999] [BEC 2000] [LEO 2002] [XIS 2005].

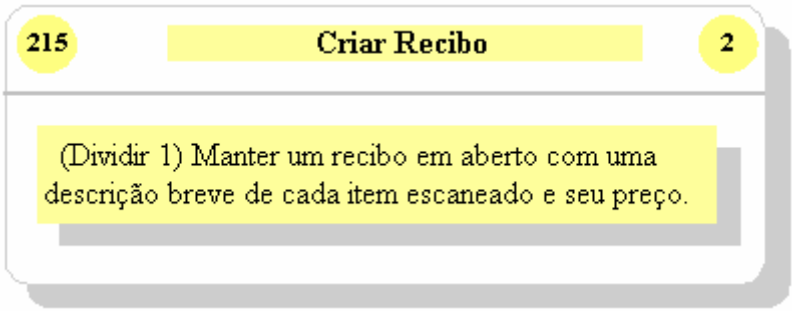


Figura 4 - Exemplo de uma *User Story* [AST 2002]

Além disso, as *User Stories* são itinerários da jornada de desenvolvimento do produto de *software*, assim como também uma descrição simples de um único aspecto do nosso sistema (como descrever uma viagem por exemplo). Assim, a descoberta das *User Stories* é um processo constante durante todo o processo da XP, sendo que elas normalmente são descritas em cartões de papel, conforme o *layout* exibido nas Figuras 4 e 5.

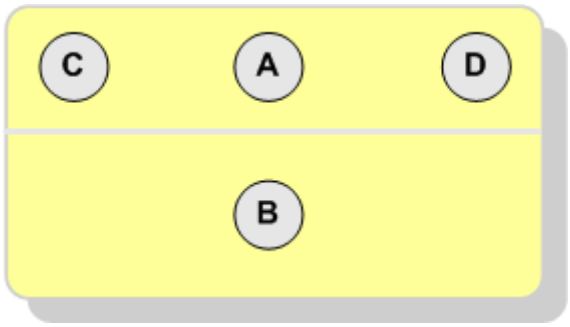


Figura 5 - *Layout* de um cartão *User Story*

A seguir, podemos ver os quatro passos necessários para criar uma *User Story*:

A) Comece com o objetivo do sistema (no centro, na parte superior do cartão), com duas ou três palavras e começando por um verbo;

B) Escreva uma única etapa de uma a três sentenças em um cartão de índice (chamamos cartão de índice porque na posição C o cartão recebe uma numeração), sendo que cada etapa capturada em um cartão de índice é uma *User Story*;

C) Atribua um número ao cartão;

D) Nesta posição defina uma estimativa para a implementação da *User Story*, sendo que uma estimativa é definida em *Story Points* (pontos de história), assim, cada *Story Point* corresponde a uma semana de quarenta horas, também chamada de semana ideal. Se a *User Story* estiver entre uma ou três *Story Points* escreva na posição D o número 1, 2 ou 3. Já, se a *User Story* for menor que uma *Story Point*, escreva ZERO na posição D.

Então, caso uma *User Story* for estimada em mais de três *Story Points*, ela é considerada grande demais para ser programada em uma iteração curta, e chamada de "Épico". Portanto, é necessário dividi-la e escrever "DIVIDIR" na posição D.

Além disso, *User Stories* que levem menos de uma semana para serem realizadas podem complicar o planejamento dos *releases*, mas são ótimos elementos para as iterações complicadas por férias ou membros novos na equipe; sendo que esses tipos de *User Stories* também fornecem vitórias de desenvolvimento rápido dentro de um *release*, e elevam o moral tanto no lado do desenvolvimento quanto no lado do cliente.

Além do mais, o conjunto de cartões para determinado objetivo é chamado de pilha, sendo que quando todas as *User Stories* de um determinado objetivo foram capturadas, é preciso numerar os cartões da pilha. Em outras palavras, isso acontece assim como se estivéssemos utilizando cartões de papel, em que podemos "colocar um elástico" nas *User Stories* daquele objetivo, ou seja, definimos um objetivo. Além do mais, isso mantém os objetivos separados e nos ajuda quando começarmos a planejar nossos *releases* e iterações.

Finalmente, quando tivermos todas as *User Stories* de todos os objetivos, formamos o *Deck* do projeto, o qual é a compreensão atual do cliente daquilo que deve ser realizado para criar o sistema desejado.

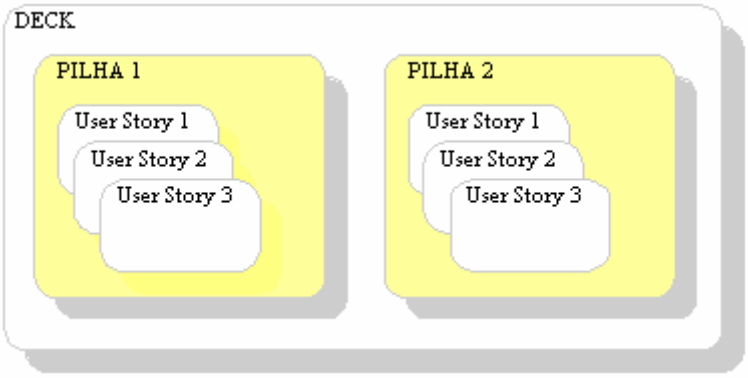


Figura 6 - Estrutura de todas as *User Stories* de um projeto

Dessa maneira, podemos notar que os cartões podem ser alterados (reescritos) e removidos para refletir uma nova compreensão do sistema, sendo que as *User Stories* serão decompostas em tarefas, que darão origem a um cronograma com iterações, entregando as *releases* aos clientes [AST 2002].

2.4.3 Testes de aceitação

Enquanto uma *User Story* descreve um comportamento geral do sistema, um teste de aceitação é uma situação concreta que o sistema pode encontrar e que exibe aquele comportamento. Assim sendo, para cada *User Story*, deve haver pelo menos um teste de aceitação que mostre que o nosso sistema demonstrou aquele comportamento.

Além disso, um teste de aceitação tem três partes, sendo que a primeira parte são as "pré-condições" ou o "cenário" do teste, que é o número mínimo de atividades que devemos fazer antes que possamos executar o teste de aceitação. Já a segunda consiste no teste em si, e quase sempre é chamado de "operação". Finalmente, existem as pós-condições , ou seja, aquilo que é verdadeiro após o teste estar concluído, o que é chamado de seção de "verificação".

Ainda é importante salientar que o teste de aceitação é normalmente escrito no verso do cartão *User Story*, ou pode ser ainda escrito em cartões separados, porém, nesse caso é necessário fazer um cruzamento das referências da *User Story* e do teste de aceitação. Portanto, o objetivo é encontrar um exemplo simples e concreto que reflita esse comportamento no sistema que está sendo desenvolvido.

2.4.4 Engenharia de Requisitos no processo XP

O desenvolvimento de um produto no processo de XP começa com o Jogo de Planejamento, o qual pode ser dividido em planejamento de *releases* e planejamento de iterações. Assim, durante o Jogo de Planejamento o cliente escreve as *User Stories* (requisitos), cujo esforço de implementação é estimado pelos desenvolvedores e então priorizado pelos clientes. Depois disso, os desenvolvedores dividem as *User Stories* em tarefas e atribuem uma estimativa para cada tarefa. A próxima etapa no processo de XP é o desenvolvimento, durante o qual as iterações são produzidas e distribuídas para os usuários, sendo que cada distribuição pode ter uma ou mais iterações. Portanto, as atividades de gerenciamento de requisitos afetam todas as fases de um projeto XP, conforme apresentado na Figura 7.

Além disso, os testes unitários são escritos antes do código e são realizados durante todas as etapas do desenvolvimento. Por sua vez, os testes de aceitação são usados para validar a completude das *User Stories*.

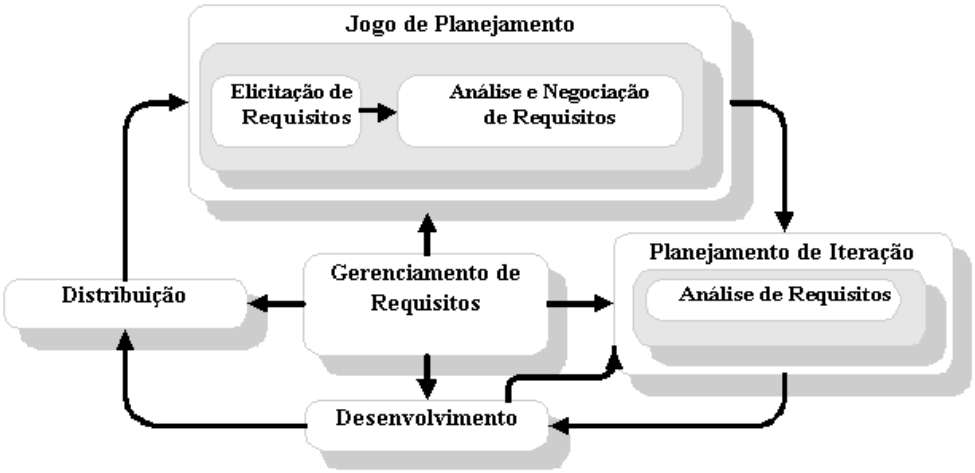


Figura 7 - Engenharia de Requisitos no processo XP

2.4.5 Ciclo de vida de uma *User Story*

Para mostrar o ciclo de vida de uma *User Story*, usamos o diagrama de transição da Figura 8, que ilustra o ciclo de vida que ela deve seguir.

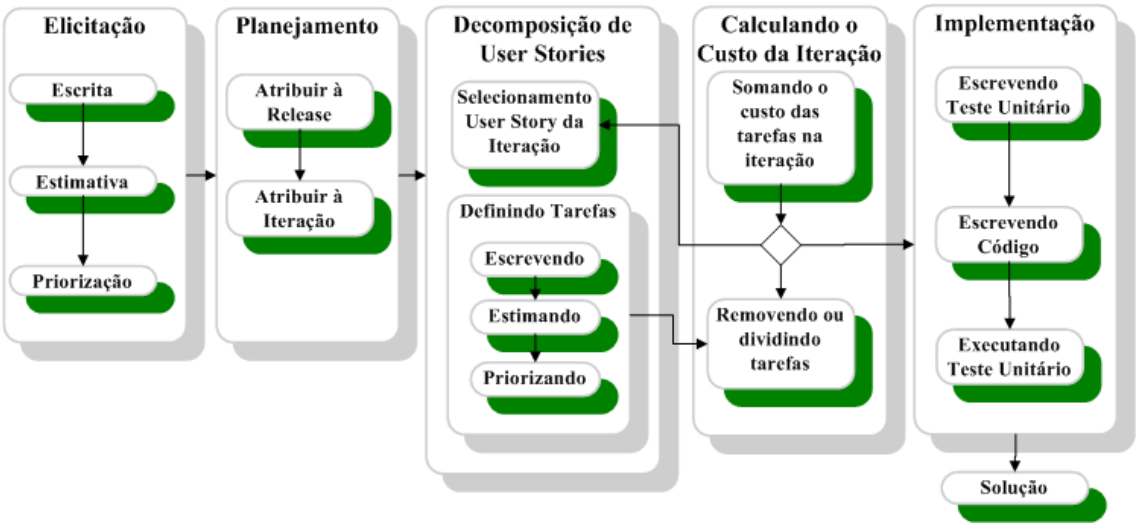


Figura 8 - Diagrama de transição ilustrando o ciclo de vida de uma *User Story*

2.4.6 *Workflow* de uma iteração em *Extreme Programming*

Cada iteração deve seguir o *workflow* apresentado na Figura 9, o que significa que podemos ter uma ou mais iterações até chegar a um *release* final. Ou seja, em cada iteração os requisitos são selecionados, os desenvolvedores estruturam e estimam as tarefas (cada tarefa representa uma *User Story*), posteriormente o usuário prioriza cada tarefa, e, finalmente, a solução é implementada e testada, gerando um *release*.

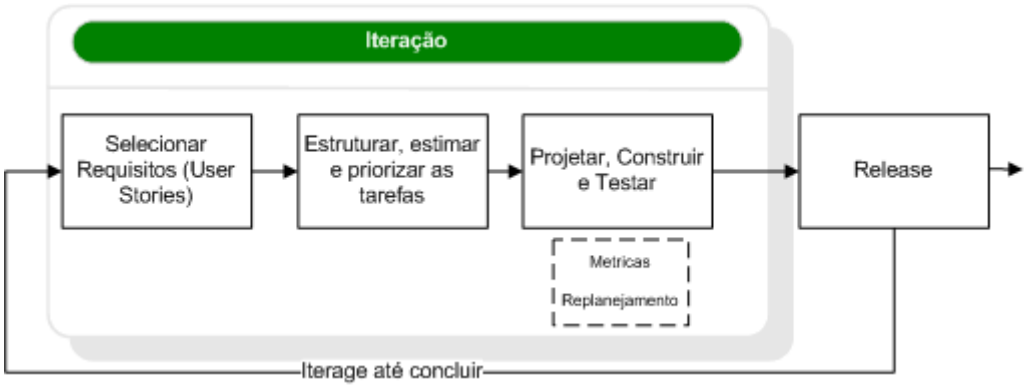


Figura 9 - *Workflow* de uma iteração em XP [CXO 2001]

2.5 Mapas conceituais

Os Mapas Conceituais (MCs) foram desenvolvidos nos anos 70 por Joe Novak [NOV 1984] e sua equipe de pesquisa na Universidade de Cornell como meio de ajuda na determinação do progresso de estudantes em seu conhecimento de Ciências.

Como definição, podemos dizer que Mapas Conceituais são representações gráficas bidimensionais do conhecimento de uma pessoa (ou grupo de pessoas) sobre um assunto, sendo que um Mapa Conceitual consiste em um conjunto de conceitos, definidos por Novak (*ibid*) como regularidades em eventos ou objetos, ou registros de eventos ou objetos, designados por um nome. Além disso, eles são construídos de forma que os conceitos e suas inter-relações sejam claros e evidentes, sendo que os conceitos são geralmente incluídos dentro de círculos ou caixas, e ligados por arcos que especificam a relação entre eles. Na maioria das vezes, o nome dos conceitos é constituído de um único termo, mas nomes compostos e mesmo símbolos como "+" ou "%" também podem ser utilizados.

Por sua vez, os conjuntos "conceito-relação-conceito" formam proposições, que são afirmações significativas sobre os objetos ou eventos envolvidos, e as quais definem as unidades semânticas, ou unidades de conhecimento do mapa, sendo que em um Mapa Conceitual bem construído, as proposições devem fazer sentido, mesmo quando lidas separadamente.

A Figura 10 ilustra um Mapa Conceitual sobre Mapas Conceituais, sendo que uma característica importante deste mapa é que os conceitos são representados de forma hierárquica, com os mais inclusivos, ou genéricos, localizados na parte superior do mapa e os mais específicos localizados na parte inferior. Dessa forma, o eixo vertical do mapa define uma estrutura hierárquica para os conceitos, em que o conceito "Mapas Conceituais" no alto da hierarquia define o domínio do conhecimento ao qual o mapa se refere, e no qual o conjunto "Mapas Conceituais -> representam -> Conhecimento Organizado" é um exemplo de proposição. Ainda é bom salientar que, em geral, proposições são lidas de cima para baixo, a menos que setas sejam usadas para indicar uma direção contrária [CAR 2005].

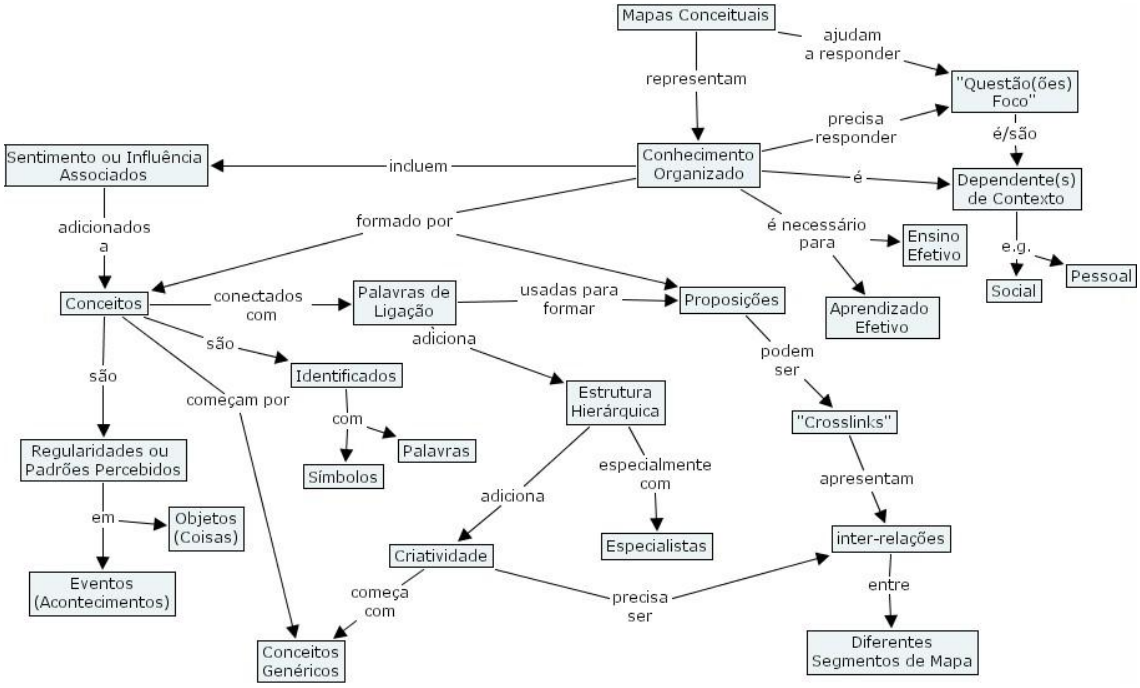


Figura 10 - Mapa conceitual sobre Mapas Conceituais [CAR 2005]

2.5.1 Características dos Mapas Conceituais

Os Mapas Conceituais têm certas características particulares que os tornam atraentes para ferramentas de gerência de projetos, tais como:

1. MCs têm estrutura: por definição, conceitos mais gerais são apresentados ao alto, na parte superior dos mapas e conceitos mais específicos na parte inferior. Semelhante a uma estrutura de desmembramento de trabalho na área de gestão de projetos (EDT ou *Work Breakdown Structure*) [VER 2000];
2. MCs são baseados em proposições: cada par de conceitos unidos por uma frase de conexão forma uma "unidade de conhecimento", que é uma estrutura baseada em proposições que distingue o MC de outras ferramentas tais como *Mind Mapping* e *Decision Explorer* [BEU 2003] e [CAB 2000], além de estabelecer um relacionamento semântico entre os conceitos;
3. MCs têm um contexto: um MC é uma representação do conhecimento sobre um determinado assunto, como uma *User Story*. De tal modo que, todos os conceitos e frases de conexão devem ser interpretados dentro desse contexto.

Podem-se observar as seguintes características em MCs bem construídos:

1. Os conceitos e as frases de ligação são os mais curtos possíveis, preferencialmente compostos por uma única palavra;
2. Cada par de conceitos ligados por uma frase de conexão forma uma proposição autônoma. Isto é, a proposição pode ser lida independente do mapa e ainda assim fazer sentido;

3. A estrutura é hierárquica e o conceito usado como raiz do mapa é um bom indicativo do seu tópico [CAR 2005].

2.5.2 A falta de formalismo dos Mapas Conceituais

A estrutura de um Mapa Conceitual (MC) depende do seu contexto, então, conseqüentemente, mesmo mapas construídos com conceitos similares podem apresentar grandes variações em função do contexto a que se referem. Assim sendo, o poder dos MCs está em sua habilidade de medir o conhecimento de uma pessoa sobre um determinado assunto dentro de um determinado contexto, e, como conseqüência, MCs construídos por pessoas diferentes sobre um mesmo tópico são possivelmente diferentes, pois cada mapa representa o conhecimento pessoal de seus autores. Dessa forma, não podemos nos referir a um MC como o único correto para um determinado assunto, uma vez que há inúmeras representações corretas possíveis.

Já no âmbito da educação, as técnicas de Mapas Conceituais ajudam pessoas de todas as idades em diversas áreas do conhecimento, pois, quando conceitos e relações são escolhidos com cuidado, os mapas tornam-se ferramentas poderosas para observar a nuance dos significados. Em outras palavras, seu rico potencial de expressão vem da capacidade de cada mapa em permitir ao seu criador o uso de um número virtualmente ilimitado de frases de ligação para definir o significado da relação entre conceitos.

2.5.2.1 Princípios organizacionais dos Mapas Conceituais

Como já foi mencionado, o método de Mapas Conceituais podem ser aplicados em diferentes áreas de conhecimento. Entretanto, neste trabalho a ênfase não será de cunho educacional, mas sim focado no relacionamento entre usuários e desenvolvedores no processo de elicitação de requisitos.

Assim, a técnica de Mapas Conceituais mostra tanto para o desenvolvedor quanto para o usuário a possibilidade de uma organização cognitiva, visto que podem indicar com precisão os conceitos chaves primordiais necessários no sistema desejado.

Além disso, quando ocorre a aprendizagem significativa, os conceitos sofrem uma transformação, em vista de sucessivas interações. Portanto, a teoria de Ausubel [AUS 1986] identifica que é mais fácil se desenvolver conceitos quando se parte de elementos mais gerais, até chegar aos mais específicos, como também diferenciar detalhes de especificidade, uma vez que esses refinamentos são detalhados com os princípios de diferenciação progressiva e reconciliação integrativa.

Já segundo Moreira [MOR 1982], o princípio da diferenciação progressiva identifica que as idéias mais gerais e inclusivas de um determinado assunto são apresentadas, para posteriormente serem diferenciadas, destacando as necessárias especificações. Assim, quando o ser humano é exposto a um novo campo de conhecimento, a consciência dos indivíduos é representada dessa maneira.

Então, a reconciliação integrativa é o princípio, cujo material instrucional deve explorar o relacionamento de idéias, a fim de identificar as semelhanças e as diferenças significativas, como também reconciliar discrepâncias reais ou aparentes.

Dessa maneira, a reconciliação integrativa deve organizar os conceitos apontados, descendo e subindo nas estruturas hierárquicas, à proporção que novas informações são apresentadas, e, assim, inicia-se com conceitos mais gerais passando para os subordinados, conhecidos como intermediários, que são relacionados finalmente com os mais específicos. Posteriormente, pode-se fazer o caminho inverso, principalmente através de exemplos, até chegar ao conceito mais geral do mapa.

A Figura 11, para Moreira [MOR 1987], representa o modelo ausubeliano de diferenciação progressiva e reconciliação integrativa.

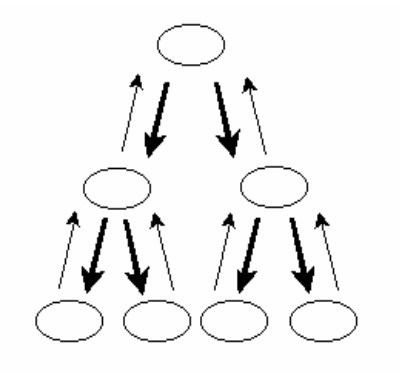


Figura 11 - Representação esquemática dos princípios do modelo ausubeliano [1]

Além do mais, a Figura 11 ressalta o processo como sendo bidimensional, pois é possível navegar pelos conceitos mais gerais até chegar aos mais específicos, ou vice-versa, sendo que, ao fazer esses dois estilos de navegação, comprovam-se os princípios da teoria de David Ausubel [AUS 1986] [MOR 1982]; caracterizando as linhas fortes como a diferenciação progressiva, e as linhas fracas como a reconciliação integrativa.

As ênfases de diferenciação progressiva e de reconciliação integrativa podem ser notadas no processo de elicitação de requisitos, sendo que esse enfoque é salientado porque as ênfases citadas representam uma forma de comunicação entre desenvolvedor e usuário, pois possibilita a distinção de elementos fundamentais para o desenvolvimento do sistema, bem como os seus relacionamentos.

Também é importante salientar que a capacidade de relacionar idéias novas com a estrutura cognitiva é utilizada como fundamento para a organização de tarefas, e torna-se necessário, então, um conhecimento sobre as funções cognitivas, o conhecimento da área abordada, e a hierarquia do conteúdo questionado, a fim de analisar os princípios de diferenciação progressiva e reconciliação integrativa.

Finalmente, portanto, os Mapas Conceituais são instrumentos facilitadores no processo de elicitação, porque possibilitam uma visão gráfica de um determinado domínio de problema e seus relacionamentos. De acordo com Ontoria [ONT 1993], são de fato um bom instrumento para detectar com rapidez a quantidade e a qualidade das informações em determinado momento, uma vez que se evidencia, com grande clareza, o número de conceitos relevantes para os usuários, assim como também a estruturação dos mesmos.

---

[1] Representação do modelo ausubeliano extraído de [MOR 1987 – pg. 27].

2.6 *CmapTools*

O *software CmapTools* foi desenvolvido pelo IHMC[2], e é uma ferramenta que, segundo Cañas [CAB 2002] , permite que o usuário construa, navegue e até mesmo compartilhe o conhecimento que foi expresso na construção dos mapas.

Na verdade, esse *software* apresenta duas ferramentas que complementam o desenvolvimento de Mapas Conceituais, as quais são denominadas de *CmapTools* e *CmapServer*; sendo que a primeira é o local de elaboração dos Mapas Conceituais, e a segunda permite o compartilhamento dos mapas com outros usuários, sendo que ambas podem ser adquiridas no *site* de *download* do *sofware*[3], embora a *CmapTools* seja uma ferramenta gratuita e a *CmapServer* não.

Além disso, a *CmapTools* possibilita que os mapas sejam elaborados individualmente ou coletivamente. Dessa forma, os mapas elaborados ficam organizados porque o *software* armazena dentro de um diretório, denominado de "local", vários projetos, sendo que, após a criação de um determinado projeto, existe a possibilidade de vincular novos mapas.

Todo mapa elaborado pode ser simplesmente consultado, como também alterado. Entretanto, se o usuário desejar alterar um mapa que já tiver sido fechado, ele não conseguirá efetuar as modificações diretamente, porque o trabalho estará vinculado a um modo de visualização. Dessa forma, cabe ao usuário alterar o modo de visualização para o modo de edição.

O Mapa Conceitual confeccionado no *CmapTools* apresentará um conjunto de proposições, ou seja, conceitos e palavras de ligação; sendo que as palavras chaves, ou conceitos, são salientados por um retângulo e suas respectivas ligações são delimitadas no momento em que se efetiva a união entre dois conceitos, de maneira que o usuário cria facilmente as proposições através da seleção dos dois conceitos relacionados.

Segundo Cañas [CAC 2002], a ferramenta em questão é amigável e de fácil compreensão, pois, no momento em que o indivíduo está elaborando o seu mapa, é disponibilizada a possibilidade de alterar o estilo do seu trabalho, através da modificação dos atributos (cor, estilo, e tamanho da fonte; e cor do preenchimento da caixa e das bordas). Além disso, ela faz com que a pessoa que está interagindo possa construir um Mapa Conceitual como uma ferramenta de navegação.

O *CmapTools* propicia uma visão de modelo de conhecimento, porque possibilita que os usuários enriqueçam os seus MCs por mídias, como URLs, vídeos, sons, e mapas. Por exemplo, para sinalizar que o Mapa Conceitual apresenta uma mídia anexada em um determinado conceito, criam-se ícones junto com os conceitos.

Portanto, quando o usuário concluir a confecção do Mapa Conceitual, ele tem a oportunidade de exportar o seu trabalho para diferentes formatos, como por exemplo, um documento *web*, uma figura ou um arquivo XML [BEU 2003] [CAB 2000].

CmapTools é baseado em uma arquitetura flexível modular sobre um módulo central chamado "*Core*". Além disso, essa arquitetura permite o desenvolvimento de novos

---

[2] http://www.ihmc.us

[3] http://cmap.ihmc.us/download

módulos, ou seja, é possível utilizar todo o potencial oferecido pelo *CmapTools* e estender a ferramenta com novas funcionalidades.

A arquitetura do *CmapTools* proporciona uma plataforma rica para pesquisas sobre o uso de ferramentas de Mapas Conceituais, sendo que dentre esses módulos podemos citar os seguintes:

- o "*suggester*", que ajuda o usuário na construção de novos Mapas Conceituais sugerindo novos conceitos, proposições, recursos, e mídias; e que guia o usuário para a construção de bons mapas;

- o "*presentation*", recurso que possibilita apresentar o Mapa Conceitual em forma de *slides*, permitindo ao usuário criar o conteúdo e a seqüência de *slides* baseando-se em um Mapa Conceitual;

- o "*recorder*", que após ativado grava cada etapa de construção do Mapa Conceitual e posteriormente permite ao usuário visualizar a seqüência armazenada;

- o "*Wordnet*", que elucida o sentido ambíguo das palavras em MCs, tanto em conceitos como em frases de ligação;

- e outros módulos já existentes no conjunto de ferramentas continuam ainda sendo realçados.

2.7 Gerência de projetos

É importante fazer uma revisão de alguns conceitos sobre gerência de projetos, pois os requisitos de um projeto de *software* representados (com a utilização de *Extreme Programming* e MCs) deverão possibilitar a geração de um cronograma inicial, que posteriormente será controlado e alterado pela equipe, possibilitando realizar estimativas mais precisas, estimar custos, alocação de recursos, ou seja, refinar a gestão do projeto [NET 2002] [VAR 1998] [VER 2000].

2.7.1 Projeto

Segundo Vargas, “projeto é um empreendimento não repetitivo, caracterizado por uma seqüência clara e lógica de eventos, com início, meio e fim, que se destina a atingir um objetivo claro e definido, sendo conduzido por pessoas dentro de parâmetros pré-definidos de tempo, custo, recursos e qualidade [VAR 1998]”.

2.7.2 Estrutura de desmembramento de trabalho

Mesmo que tenhamos um grande entendimento do projeto para equilibrar custo, cronograma e qualidade, será necessário desmembrá-lo para que se possamos entender o projeto como um todo através do entendimento de suas partes.

Sendo assim, a estrutura de desmembramento de trabalho (EDT, ou em inglês WBS - *Work Breakdown Structure*) é um método para desmembrar um projeto em seus

componentes e partes, e, por isso, uma das técnicas mais importantes usadas na gestão de projetos.

Além disso, a estrutura de desmembramento de trabalho identifica todas as tarefas de um projeto; é às vezes chamada simplesmente de lista de tarefas; transforma o projeto em muitas tarefas pequenas e gerenciáveis; usa os resultados da definição do projeto e do gerenciamento dos riscos, e, ainda, identifica as tarefas que são a base de todo o planejamento subseqüente.

Por exemplo, a EDT na forma de tópicos (como mostra a Figura 12) para projetar e colocar uma nova grama com um sistema de *sprinklers* para a irrigação, rodeada por uma cerca nova, exige algumas tarefas diferentes. Assim, o gráfico de EDT cria uma figura que facilita o entendimento de todas as partes de um projeto, mas, entretanto, a EDT em tópicos é mais prática porque podemos listar centenas de tarefas.

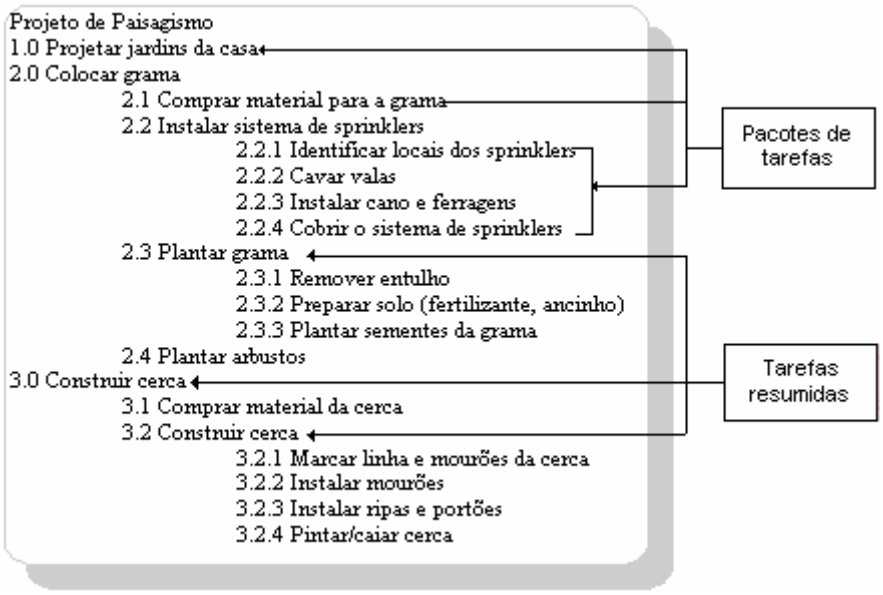


Figura 12 - Estrutura de desmembramento de trabalho na forma de tópicos

Assim, podemos notar que a EDT desmembra todo o trabalho do projeto em tarefas separadas (as tarefas também podem ser chamadas de atividades), sendo que há dois tipos de tarefas em uma EDT: tarefas de resumo e pacotes de trabalhos.

Portanto, no exemplo visto na Figura 12, "Instalar o sistema de *sprinklers*" para um gramado é uma tarefa de resumo, já que inclui diversas tarefas subordinadas, sendo que a instalação do sistema de *sprinkler* pode ser composta de tarefas distintas e subordinadas, tal como as valas cavadas ou a instalação dos canos. Dessa forma, executando esse pacote de tarefas simples, podemos completar uma tarefa de resumo (Figura 12). No entanto, uma tarefa de resumo na verdade não é executada, ela é um resumo dos pacotes de trabalhos subordinados, sendo que, por sua vez, os pacotes de trabalho é que são executados de verdade. Sendo assim, o entendimento da relação entre as tarefas de resumo e os pacotes de trabalhos é fundamental na construção de uma boa EDT.

2.8 Web Semântica

O grande aumento da utilização da *web* tem desencadeado o surgimento de novas tecnologias. Com isso, procura-se melhorar a forma de representar e posteriormente manipular as informações contidas nessa rede para atender seus usuários de uma forma mais eficiente, e, assim, a *web* semântica surge como um novo conceito para organização de dados, tendo como objetivo a busca de um padrão capaz de suportar o entendimento semântico das informações. Além disso, a *web* semântica proporciona uma estrutura comum que permite o compartilhamento e reutilização dos dados entre aplicações, empresas e grupos de usuários [GON 2005] e [SWE 2005].

Além da informação, os documentos *web* possuem ainda a estrutura do documento e as informações referentes ao seu contexto, sendo que a descrição deste conceito é feita por meio do conceito de metadados[4]. Entretanto, a padronização dos dados não é o bastante para a *web* semântica, é preciso que se tenham fontes comuns de informações, sendo um vocabulário de dados organizado para cada domínio de aplicação. Por isso, surge a necessidade de compreender também o conceito de ontologias, sendo que esses conceitos serão apresentados a seguir.

2.8.1 RDF (*Resource Description Framework*)

Como os documentos *web* precisam de metadados para expressar o seu valor semântico, é preciso utilizar uma linguagem que faça a descrição de tais informações, e, tem-se, assim, o padrão RDF (*Resource Description Framework*) [BEC 2004], que foi criado, sob a supervisão do W3C, para definir uma padronização da representação e do uso de metadados na *web.* Ele consiste ainda em uma infra-estrutura que habilita a codificação, a troca e o reuso de metadados, e é uma infra-estrutura que habilita a interoperabilidade através do desenvolvimento de mecanismos que suportam convenções de semânticas, sintaxe, e estrutura, permitindo o desenvolvimento de aplicações modulares.

No entanto, como RDF possui uma sintaxe XML, algumas pessoas poderão pensar em RDF com base nesta sintaxe, o que é um erro, uma vez que o seu modelo de dados deve ser entendido, ou seja, seus dados podem ser representados em XML, embora entender a sintaxe seja secundário ao entendimento do modelo de dados [RDF 2006].

Vamos utilizar exemplos sobre pessoas, os quais utilizam uma representação RDF de *vcards*[5], que, por sua vez, é utilizado para abreviar URI's[6] (é uma forma reduzida), enquanto o RDF utiliza somente URI's completas. Sendo assim, o RDF é mais bem representado em forma de diagramas de nodos e arcos, sendo que um simples *vcard* poderia ser representado como a Figura 13.

---

[4] Metadados, ou Metainformação, são dados capazes de descrever outros dados, ou seja, dizer do que se tratam, dar um significado real e plausível a um arquivo de dados, são a representação de um objeto digital (http://pt.wikipedia.org/wiki/Metadados).

[5] http://www.w3.org/TR/vcard-rdf - *RDF representation of VCARDS* habilita uma descrição comum e consistente de pessoas, utilizando uma semântica existente de *vCard* e codificando em RDF/XML

[6] URI's são *strings* utilizadas para identificar recursos na *web*, como páginas, serviços, documentos, imagens, músicas, arquivos, caixas de *e-mail*, notícias, entre muitos outros (http://pt.wikipedia.org/wiki/URI).

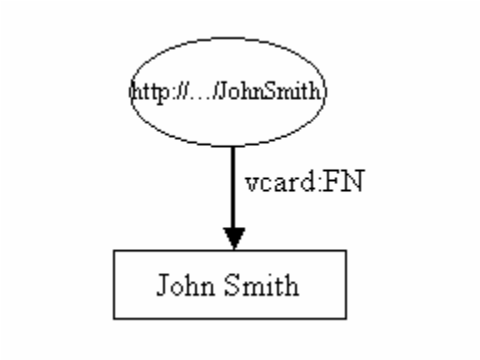


Figura 13 - Um modelo RDF representado em forma de diagramas de nodos e arcos [RDF 2006]

Um diagrama pode ser dividido em três partes:

Recurso (*Resource*)	O recurso é representado por uma elipse e identificado por uma URI (*Uniform Resource Identifier*). No nosso exemplo, o nome do recurso é *JohnSmith*. Além disso, os recursos possuem propriedades.
Propriedade (*Property*)	As propriedades são representadas por um arco com uma descrição da propriedade (*label*), as quais também são identificadas por uma URI, pois, normalmente em diagramas é conveniente utilizar um nome curto. Assim, a propriedade do nosso exemplo é o *full name* (FN) de John Smith. O que fica antes dos ":" nós chamamos de *NAMESPACE PREFIX*, que representa o *namespace*, e, após os ":" nós chamamos de *LOCAL NAME*, que representa o nome dessa *namespace*.
Literal	Cada propriedade possui um valor, nesse caso um literal, sendo que os literais são representados em retângulos com *strings* de caracteres.

Tabela 1 - As três partes de um diagrama RDF [RDF 2006]

Podemos notar ainda que cada arco (*arc*) em um modelo RDF (*RDF Model*) é chamado de *STATEMENT*, e que cada *statement* afirma um fato sobre um recurso, e possui três partes, as quais são:

Subject	É o recurso do qual o arco parte.
Predicate	É a propriedade que descreve o arco (*label*).
Object	É o recurso ou literal apontado pelo arco.

Tabela 2 - As três partes de um *STATEMENT* [RDF 2006]

Além do mais, algumas vezes o *statement* é chamado de tripla (*triple*), por causa dessas três partes.

O padrão RDF especifica ainda como representar RDF em formato XML, uma introdução mais detalhada ao RDF pode ser encontrado em http://www.w3.org/TR/rdf-primer. A Figura 14 exemplifica a sintaxe RDF.

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'>
  <rdf:Description rdf:about='http://somewhere/JohnSmith'>
    <vcard:FN>John Smith</vcard:FN>
  </rdf:Description>
</rdf:RDF>
```

Figura 14 - Exemplo da sintaxe XML do RDF

Normalmente RDF é embutido entre uma *tag* "<rdf:RDF>". No exemplo da Figura 14 o elemento RDF representa duas *namespaces* utilizadas no documento.

Há então um elemento "<rdf:Description>" que descreve o recurso da qual a URI é http://somewhere/JohnSmith. Assim, se o atributo "rdf:about" foi esquecido, esse elemento representa um nodo vazio (*blank node*).

2.8.2 Ontologias

Segundo uma visão filosófica, o termo ontologia é definido por Aristóteles como sendo uma concepção de tudo aquilo que pode "existir" ou "ser". No entanto, na área da Informática, mais precisamente em Inteligência Artificial, Grubber [GRU 1999] define uma ontologia como sendo "a especificação de uma conceitualização". Por sua vez, Guarino [GUA 1998] já estende essa definição ao afirmar que uma ontologia é na verdade "uma especificação parcial e explícita que tenta, da melhor forma possível, aproximar a estrutura de mundo definida por uma conceituação". Assim, uma ontologia é uma descrição explícita de conceitos e relações referentes a um determinado domínio, conceitualização essa que se refere ao conjunto de conceitos, relações, objetos e restrições que são definidos para um modelo semântico de algum domínio de interesse [GON 2005] [LUS 2003].

2.8.3 OWL (*Ontology Web Language*)

A OWL é uma especificação de uma linguagem para ontologias desenvolvida pela W3C[7], que faz parte da crescente lista de recomendações da W3C relacionadas ao

[7] http://www.w3.org/

desenvolvimento da *web* semântica, e que oferece mecanismos para representar explicitamente o significado dos termos e dos relacionamentos entre eles [SMI 2003].

Além disso, a OWL deriva de outras duas linguagens, a OIL (*Ontology Inference Layer*) e a DAML (*DARPA Agent Markup Language*), sendo que a OIL foi a primeira dessas linguagens, e teve como principal requisito a facilidade de adoção por parte dos desenvolvedores, servindo principalmente à comunidade ligada à *web* semântica [HOR 2000]. Então, posteriormente, as duas foram unidas, criando assim a linguagem DAML+OIL. Portanto, a Figura 15 apresenta a origem da linguagem OWL.

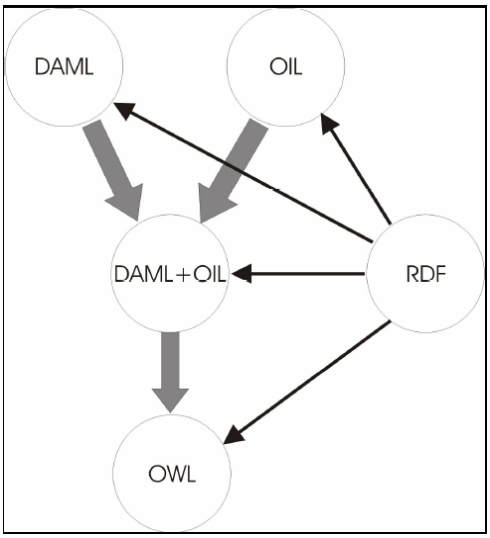


Figura 15 - Origem da linguagem OWL

Na Figura 15, pode-se observar que a linguagem OWL possui as características do RDF e um vocabulário maior que a DAML+OIL, oferecendo assim mais recursos para utilização. Com isso, a OWL é projetada para ser utilizada por aplicações que necessitam realizar o processamento do significado das informações antes de apresentá-las aos usuários [SMI 2003]. Ainda, a OWL é utilizada para representar vocabulários e os relacionamentos existentes entre as entidades desses vocabulários, o que permite a descrição de classes e propriedades.

Os recursos que a linguagem OWL oferece são divididos em três sublinguagens [SMI 2003], as quais são:

- OWL Lite – suporta usuários que precisam de uma hierarquia de classificação e funcionalidades de restrições simples. Por exemplo, a OWL Lite suporta cardinalidade, mas só permite os valores 0 e 1, assim, ela se torna mais fácil de ser implementada em uma ferramenta e faz com que a transição de outros modelos de vocabulários e taxonomias para que a OWL seja mais rápida;
- OWL DL – suporta usuários que precisam de máxima expressividade, ao mesmo tempo em que seus sistemas mantêm a completude, ou seja, a garantia que todas as conclusões serão executadas; e decidibilidade do sistema, o que significa que todos os cálculos terminarão em tempo finito. No entanto, a OWL DL inclui todos os artefatos da linguagem OWL, mas impõem restrições quanto à utilização;

- OWL Full – para usuários que desejam a máxima expressividade e a liberdade sintática do RDF, a OWL Full permite que uma ontologia aumente o significado do vocabulário (RDF ou OWL) predefinido. Entretanto, não é esperado que nenhum *software* suporte todas as características da OWL Full.

Sendo assim, a OWL Full pode ser vista como uma extensão da RDF, enquanto a OWL Lite e a OWL DL são extensões de uma visão delimitada de RDF. Cada uma dessas sublinguagens é uma extensão de seu predecessor mais simples.

2.8.4 API Jena 2

Jena é uma API para a linguagem de programação Java, desenvolvida por Brian McBride, da Hewlett-Packard (HP) [HEW 2007], que é usada na criação e manipulação de grafos RDF [JEN 2006] [GON 2005], e que possui objetos, classes para representar os modelos, recursos, propriedades e literais do RDF. Por sua vez, as interfaces representam recursos, propriedades e literais, que são chamadas de *Resource*, *Property* e *Literal*, respectivamente. Ainda em Jena, um grafo RDF é chamado de modelo e representado pela interface *Model*.

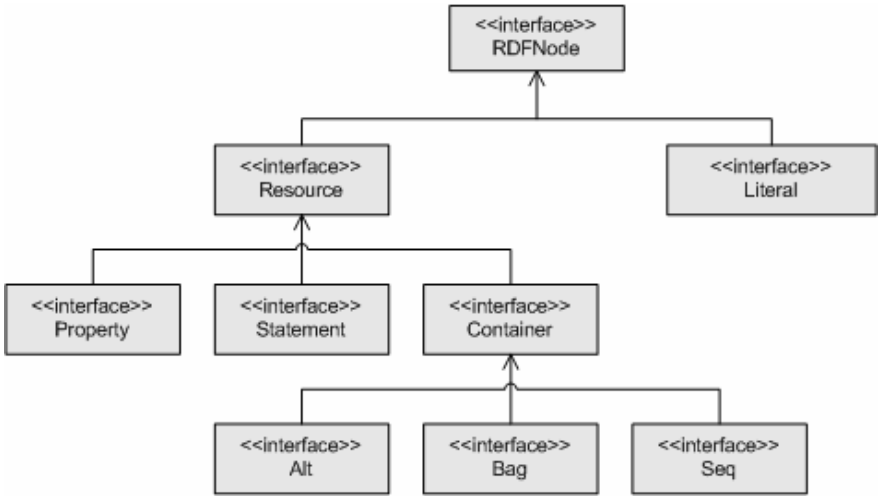


Figura 16 - Hierarquia das interfaces da API Jena [VER 2001]

As interfaces demonstradas na Figura 16 são descritas da seguinte maneira [VER 2001]:

- *RDFNode*: interface que possui a função de fazer a ligação de todos os outros elementos do RDF, a fim de formar as triplas para a criação dos elementos;
- *Resource*: interface que representa objetos que possuam uma URI;
- *Literal*: interface que representa valores usados como objetos em triplas (sujeito, predicado, objeto), e que possui métodos para converter valores para vários tipos de dados em Java, como *string*, *int* e *double*;
- *Property*: interface que representa as propriedades em uma tripla;
- *Statement*: interface que representa uma tripla;

- *Container*: interface que representa um conjunto de objetos (*Alt*, *Bag*, *Seq*)[8] em uma tripla.

2.8.4.1 Jena *Ontology*

A primeira versão da API Jena possui métodos para a manipulação de ontologias em DAM+OIL, enquanto que na segunda versão foram desenvolvidos métodos que permitem a manipulação em ontologias RDFS e OWL. Ou seja, para o suporte à manipulação de ontologias, a segunda versão da API Jena possui um pacote específico (ou *framework*), chamado de API Jena 2 *Ontology*, em que existem classes para manipulação de ontologias em RDFS, DAML+OIL e OWL. Assim, para o suporte a essas linguagens de ontologias a API possui as seguintes classes: *OntClass* e *ObjectProperty*.

Além disso, para cada uma das linguagens de ontologia existe um parâmetro que permite a construção de URI's de classes e propriedades, por sua vez, cada parâmetro possui uma sintaxe diferente. Por exemplo, no parâmetro da linguagem DAML, a URI para uma propriedade do objeto é *daml:ObjectProperty*, e na linguagem OWL é *owl:ObjectProperty*. Já no RDFS, o parâmetro é nulo, pois nesse padrão não se definem propriedades para os objetos. Portanto, cada linguagem de ontologia possui suas próprias características e é limitada ao seu modelo de ontologia, embora todas estendam a versão do modelo de classes da Jena.

Na criação de um modelo, a API Jena oferece vários pacotes que fazem a interação com o RDF, sendo que a Figura 17 demonstra os 3 módulos base para o processamento de ontologias.

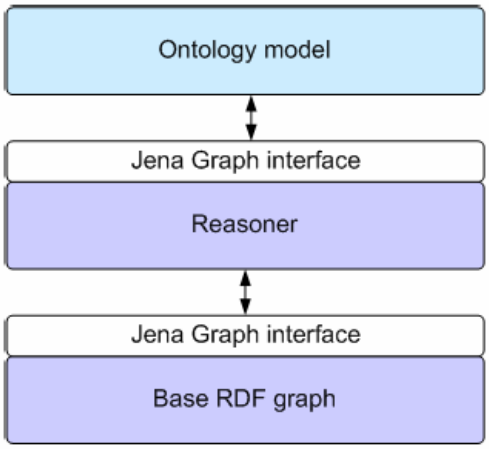


Figura 17 - Iteração de um modelo de ontologias com o RDF [JEN 2006]

[8] RDF define um tipo especial de recurso para representar coleções de coisas, as quais são chamadas de *containers*. Assim, os membros de um *container* podem ser literais ou recursos, sendo que há três tipos de *containers*: BAG, que é uma coleção não ordenada (*unordered collection*); ALT, que é uma coleção não ordenada que pretende representar alternativas; e SEQ, que é uma coleção ordenada (*ordered collection*) [RDF 2006].

O primeiro módulo é o *Ontology Model*, este contém todas as classes necessárias para trabalhar com ontologias descritas em OWL, DAML, OIL ou RDFS. Neste módulo a classe mais relevante é a *OntModel* que representa um modelo ontológico. Esta classe no entanto é uma interface.

O outro módulo presente na arquitetura é o *Reasoner*. Este permite fazer inferências sobre modelos OWL. O uso das inferências sobre modelos semânticos é permitir obter informação adicional (inferida) sobre as ontologias. Os *reasoners* de OWL do Jena funcionam aplicando regras tipo *if-then-else* sobre instancias OWL.

O OWL está definido sobre RDFS, logo o Jena usa as suas APIs de RDF apara poder manipular as ontologias. Dai a existência do terceiro módulo [ALV 2007].

2.8.4.2 Modelos de ontologias na API Jena

Um modelo de ontologia é uma extensão do modelo RDF da Jena que fornece capacidades para a manipulação de ontologias, cujos modelos são criados por meio do Jena *ModelFactory*. Assim, a maneira mais simples de se criar um modelo de ontologia é utilizando a classe *OntModel* proveniente do pacote de ontologias da Jena.

Além da capacidade simples de ler documentos, o modelo de ontologias possui algumas capacidades adicionais, como fazer o tratamento de reusabilidade e importar diferentes tipos de documentos. Por sua vez, na OWL e na DAML há uma classe individual *Ontology* que contém metadados sobre o documento, sendo que a Figura 18 mostra um exemplo da classe *Ontology* em um documento OWL.

```
<owl:Ontology rdf:about="">
   <dc:creator rdf:value="Paulo Roberto"/>
   <owl:imports rdf:resource="http://pauloroberto.com/exemplo"/>
<owl:Ontology/>
```

Figura 18 - Exemplo da classe *Ontology* [GON 2005]

Além disso, para o acesso e manipulação de ontologias, a Jena possui o pacote *com.hp.hpl.jena.ontology*, que faz a representação da ontologia em RDF, sendo que as linguagens de ontologias trabalhadas nesse pacote são OWL e DAML+OIL. Portanto, a seguir vamos descrever alguns dos recursos oferecidos por esse pacote para a manipulação de ontologias:

- ***OntResource***

Todas as classes na API *Ontology* que representam valores da ontologia têm o *OntResource* como uma superclasse, o que faz dele um bom lugar para se colocar funcionalidades compartilhadas para todas as classes.

Além disso, a interface Java *OntResource* estende a interface *Resource* da Jena RDF, e, assim um método pode aceitar um recurso, ou um *RDFNode* aceitar um *OntResource*, e conseqüentemente, algum outro valor da ontologia.

Assim, alguns dos atributos comuns dos recursos de ontologias são expressos com métodos em *OntResource* como os que são mostrados na Tabela 3:

Atributo	**Descrição**
versionInfo	É uma descrição da versão atual da ontologia.
comment	É o método que atribui um comentário/descrição desejado sobre o recurso.
label	É o elemento que especifica um nome ao recurso, e que pode conter considerações sobre o recurso.
seeAlso	É alguma URL que consulta mais informações sobre esse recurso.
isDefinedBy	É uma especialização do *seeAlso*, cuja intenção é a de fornecer uma descrição desse recurso.
sameAs	É o método que representa a equivalência entre os recursos. Por exemplo, considera-se que um recurso "*dog*" é igual ao recurso "cachorro".
differentFrom	É o método que permite mostrar que um recurso é diferente de outro. Por exemplo, se possui um recurso "Sistemas de Informação" e outro "Turismo", pode-se dizer que são diferentes.

Tabela 3 - Atributos que são expressos como métodos do *OntResource* [JEN 2006] [GON 2005]

Além do mais, podemos afirmar que para cada propriedade existem métodos padrões disponíveis, que são visualizados na Tabela 4.

Método	**Ação**
add<property>	Adiciona um valor para a propriedade.
set<property>	Remove o valor existente na propriedade e adiciona um novo valor.
list<property>	Retorna os valores da propriedade.
get<property>	Retorna o valor desta propriedade, se possuir um recurso. Já se possuir mais de um valor, seleciona apenas um.

has<property>	Retorna "verdadeiro" se existir pelo menos um valor para a propriedade dada.
remove<property>	Remove o valor de uma propriedade do recurso, mas não tem efeito se o recurso não tiver valor.

Tabela 4 - Métodos para as propriedades [JEN 2006] [GON 2005]

No *ontResource* existem ainda outros métodos para utilidade geral. Por exemplo, para procurar quais valores um recurso possui para uma propriedade específica, pode-se chamar o método *getCardinality(Property p)*, já para excluir um recurso de uma ontologia, chama-se o método *remove()*, sendo que esse método remove cada sentença que utiliza esse recurso, como um sujeito ou objeto de uma sentença por exemplo. Por outro lado, para alterar ou buscar o valor de um recurso dado, utiliza-se o método *setPropertyValue(Property p, RDFNode valor)* ou *getPropertyValue(Property p)*, respectivamente. Similarmente, os valores das propriedades podem ser adicionados e removidos.

- ***OntClass***

Uma classe simples é representada na Jena por um objeto *OntClass*, pois a representação de uma classe permite a criação de várias instâncias que apresentam as características dessa classe. Assim, como mostra o exemplo da Figura 19 (considerando que "m" é um objeto do tipo *OntModel*), uma classe pode ser a conversão de um recurso RDF.

```
String art = "http://www.paulo.com/owl/ontologia/artista/#";
Resource recurso = m.getResource( art + "Artista" );
OntClass artista = (OntClass) recurso.as( OntClass.class );
```

Figura 19 - Exemplo *OntClass* [GON 2005]

Entretanto, isso ainda pode ser simplificado fazendo um chamado ao método *getOntClass()* no modelo de ontologia, como é apresentado na Figura 20.

```
String art = "http://www.paulo.com/owl/ontologia/artista/#";
OntClass artista = m.getOntClass( art + "Artista" );
```

Figura 20 - Exemplo do método *getOntClass* [GON 2005]

Além disso, para a criação de uma classe com um recurso que não existe, é possível utilizar o método *createClass()*, como no exemplo mostrado da Figura 21.

```
String art = "http://www.paulo.com/owl/ontologia/artista/#";
OntClass novo = m.createClass( art + "NovoRecurso" );
```

Figura 21 - Exemplo do método *createClass* [GON 2005]

Dessa maneira, depois de criado um objeto *OntClass*, pode-se chamar os métodos definidos nessa classe, sendo que seus atributos são similares aos atributos do *OntResource*, com uma coleção de métodos para modificar (*set*), adicionar (*add*), listar (*list*) e remover (*remove*) valores. Portanto, na Tabela 5, podemos observar alguns atributos de um objeto *OntClass*.

Atributo	Descrição
subClass	Uma subclasse de uma classe são aquelas declaradas com o *subClassOf*.
superClass	Uma superclasse é aquela que possui subclasse, ou seja, é a superclasse de uma subclasse.
equivalentClass	Uma classe que representa o mesmo conceito que outra classe. Por exemplo, uma classe "Estado", pode ser definida como uma classe "UF".
disjointWith	Representa classes disjuntas, por exemplo, as classes "animal" e "fruta".

Tabela 5 - Atributos de *OntClass* [JEN 2006] [GON 2005]

- ***OntModel***

O *OntModel* é um modelo de ontologia visualizado pela Jena, o qual oferece recursos para os tipos de objetos esperados em uma ontologia, tais como: classes, propriedades e recursos. Assim, a criação de um modelo simples é feita como segue:

```
OntModel modelo = ModelFactory.createOntologyModel.createOntologyModel();
```

Quando um modelo é criado dessa forma, passa-se a usar a linguagem OWL como linguagem padrão. Ainda, para criar um modelo de ontologia em uma linguagem em particular, mas deixando todos os valores padrões, é só especificar uma URI da linguagem de ontologia específica, sendo que as URI's das linguagens podem ser visualizadas na Tabela 6:

Linguagem de ontologia	URI
RDFS	http://www.w3.org/2000/01/rdf-schema#
DAML+OIL	http://www.daml.org/2001/03/daml+oil#
OWL Full	http://www.w3.org/2002/07/owl#
OWL DL	http://www.w3.org/TR/owl-features/#term_OWLDL
OWL Lite	http://www.w3.org/TR/owl-features/#term_OWLLite

Tabela 6 - URI's das linguagens de ontologias suportadas pela Jena [JEN 2006]

Além disso, para a criação de um modelo simples de ontologia na linguagem DAML, utiliza-se:

```
OntModel modelo =
ModelFactory.createOntologyModel(profileRegistry.DAML_LANG);
```

Sendo assim, para a criação na qual se deseja o controle completo sobre o modelo de ontologia, que inclui a linguagem em uso, a máquina de inferência e os meios de manipulação de documento, utiliza-se a primitiva *OntModelSpec*[9].

Portanto, a criação de um modelo com a especificação dada chama o *ModelFactory*, como segue:

```
OntModel modelo = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM,
null);
```

● ***OntProperty***

Uma propriedade em um modelo de ontologia é uma extensão da classe *Property* em RDF, sendo que a superclasse comum para representação de propriedades de ontologias em Jena é *OntProperty*. Isso significa que esse elemento refere-se às propriedades de uma classe, ou seja, às características ou relacionamentos entre os objetos. Portanto, uma propriedade, além de especificar a classe a qual ela pertence, ou seja, a classe que contém tal característica ou relacionamento, também apresenta o tipo de seu conteúdo. Por sua vez, os atributos de uma *OntProperty* podem ser visualizados na Tabela 7.

[9] http://jena.sourceforge.net/javadoc/com/hp/hpl/jena/ontology/OntModelSpec.html

Atributo	Descrição
subProperty	É uma subpropriedade de uma propriedade, que é declarada por meio do *subPropertyOf* de outra propriedade. Por exemplo, se "P1" é subpropriedade de "P", e se tem "A p1 B", então pode-se dizer que "A p B" é verdade.
superProperty	É uma superpropriedade de uma propriedade.
Domain	É o atributo que adiciona uma classe ou várias classes para formar o domínio da propriedade, sendo que, múltiplos domínios são interpretados como conjuntos.
range	É o atributo que define o tipo de valor que a propriedade pode assumir.
equivalentProperty	É o atributo que mostra que uma propriedade tem o mesmo conceito que outra, assim como já foi citado no exemplo de classes equivalentes (*equivalentClass*).
Inverse	É o atributo que mostra que uma propriedade é inversa à outra. Por exemplo, se "p1" é inverso de "p", e tem-se "A p1 B", então pode-se dizer que "B p A" é verdade.

Tabela 7 - Atributos de *OntProperty* [JEN 2006] [GON 2005]

● ***ObjectProperty***

É o objeto que representa o tipo do valor a qual a propriedade vai assumir, e que é utilizado para representar o relacionamento de uma classe com outra. Além do mais, é ele que apresenta como valor a instância de outra classe.

● ***DatatypeProperty***

É semelhante à especificação anterior (*ObjectProperty*), mas, no entanto, possui a restrição da propriedade se relacionar com apenas um tipo de dados específico. Além disso, ela deve assumir um valor já definido pelo XML *Schema*, sendo que alguns desses valores são: *integer*, *string*, *float,* e *boolean*.

2.9 UML-MC

UML [BOO 1999] [BOO 2000] é uma linguagem de modelagem padronizada pela OMG [OMG 2007], em 1997, cujo objetivo é especificar, visualizar, construir e documentar todas as informações necessárias para o desenvolvimento de sistemas de

software orientado a objetos. Além disso, a UML fornece um conjunto de diagramas compostos por elementos e relacionamentos, que permitem criar modelos gerais para um sistema de *software*. Porém, muitas vezes não podemos comunicar de forma clara e precisa todos os detalhes de um sistema, utilizando apenas essa notação básica, portanto, para resolver esse problema, a UML dispõe de três mecanismos de extensão (*stereotypes*, *tagged values* e *constraints*) que permitem estender a linguagem de forma controlada, para que a modelagem de um sistema possa ser mais detalhada.

UML-MC utiliza os mecanismos de extensão fornecidos pela UML para representar os Mapas Conceituais, sendo que os mecanismos de extensão da UML são recursos disponibilizados pela linguagem, e permitem que ela seja adaptada para atender a domínios específicos. Em outras palavras, essa notação auxilia a modelagem de sistemas baseados em Mapas Conceituais utilizando uma notação estendida, a fim de não precisar se preocupar com detalhes de implementação necessários para a criação de Mapas Conceituais [ROB 2003].

A seguir, vamos descrever as principais características de alguns dos mecanismos de extensão (*stereotypes*) da notação UML-MC:

2.9.1 O estereótipo <<nodo_pai>>

Intenção: representar o conceito principal de um Mapa Conceitual.

Problema: expressar o nome do ambiente e possuir dependentes que indicarão informações correspondentes a ele.

Solução: definir uma nova classe que represente o conceito principal de um Mapa Conceitual, que aparece sempre no topo da estrutura de cada Mapa Conceitual. Ou seja, essa classe deverá possuir subclasses que indicarão informações correspondentes a ele.

Participantes e colaboradores: <<nodo_pai>>.

Implementação: a cabeça da flecha sob a classe AulaNet significa que aquelas classes que apontam para ela, dela derivam. Além disso, isso significa que é uma classe abstrata, ou seja, uma classe utilizada para definir uma interface para as classes que derivam dela.

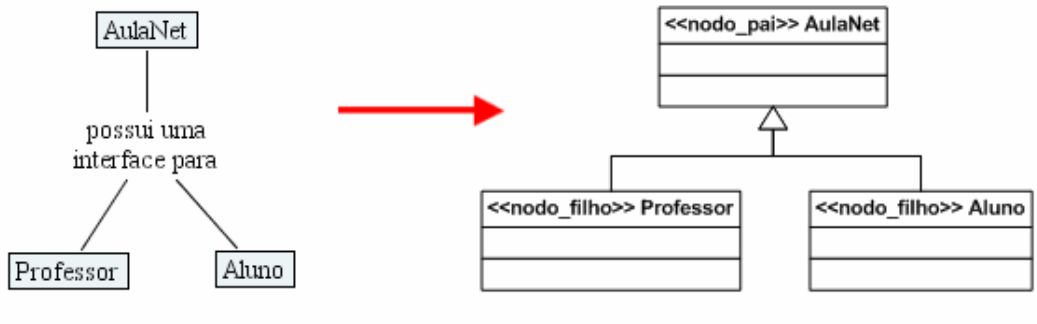


Figura 22 - Utilização do estereótipo <<nodo_pai>> (relacionamento "é-um")

2.9.2 O estereótipo <<nodo_filho>>

Intenção: esclarecer algo sobre o conceito, quando for necessário exemplificar o conceito a que está relacionado.

Problema: ser a informação mais específica que existe no Mapa Conceitual.

Solução: o mesmo apresentado em <<nodo_pai>>.

Participantes e colaboradores: um <<nodo_filho>> que sempre é subordinado a um <<nodo_pai>>, a um <<nodo_primo>> ou a um <<nodo_pai&filho>>.

Implementação: a mesma apresentada em <<nodo_pai>>.

2.9.3 O estereótipo <<nodo_pai&filho>>

Intenção: adicionar alguma informação a respeito de um conceito mais geral, que pode ser de um <<nodo_pai>>, de um <<nodo_primo>> ou de outro <<nodo_pai&filho>>.

Problema: ocupar um nível intermediário do Mapa Conceitual, sem representar o conceito principal do Mapa Conceitual, nem o conceito mais específico.

Solução: criar um <<nodo_pai&filho>> que represente um nível intermediário do Mapa Conceitual.

Participantes e colaboradores: <<nodo_pai&filho>> que pode adicionar alguma informação a respeito de um conceito mais geral, que pode ser de um <<nodo_pai>>, de um <<nodo_primo>> ou de outro <<nodo_pai&filho>>. Além disso, o nodo <<nodo_pai&filho>> também possui conceitos dependentes (<<nodo_filho>> ou outro <<nodo_pai&filho>>) que expressam informações a seu respeito.

Implementação: para apresentar a implementação, vamos analisar um exemplo, o conceito "Comunicação", que é um <<nodo_pai&filho>>, uma vez que especifica alguma informação sobre conceito "Mecanismos", assim como também sobre os conceitos "Grupo de Discussão" e "Debate", que exemplificam tipos de mecanismos de comunicação existentes.

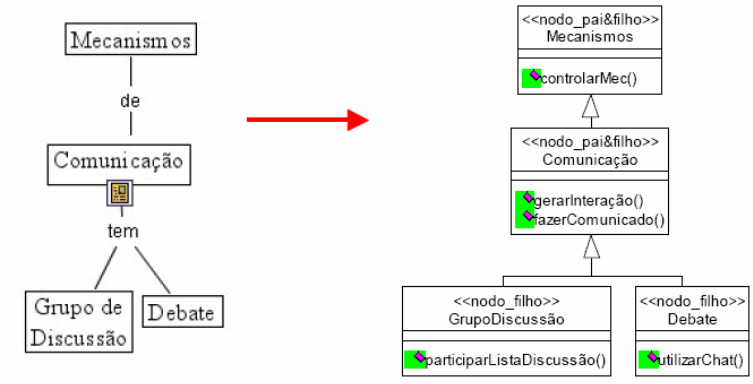


Figura 23 - Utilização do estereótipo <<nodo_pai&filho>>

2.9.4 O estereótipo <<nodo_primo>>

Intenção: um <<nodo_primo>> é uma especificação de um conceito (<<nodo_pai>> ou <<nodo_pai&filho>>) e pode ou não ser um conceito geral de outro, isto é, não necessariamente é fornecida alguma informação sobre o <<nodo_primo>>.

Problema: criar um <<nodo_primo>> somente quando conceitos encontram-se em um mesmo nível e relacionam-se entre si.

Solução: um <<nodo_primo>> sempre está ligado a outro <<nodo_primo>> por meio de uma associação unidirecional, pois um informa ou recebe alguma informação sobre outro.

Participantes e colaboradores: <<nodo_primo>>, outro <<nodo_primo>>, <<nodo_pai>> e <<nodo_pai&filho>>.

Implementação: para apresentar a implementação, vamos analisar um exemplo, o <<nodo_primo>> "Administrador", que se relaciona em um mesmo nível e é uma especificação do conceito AulaNet. Assim, os nodos <<nodo_primo>> "Professor" e "Aluno" relacionam-se com o <<nodo_pai&filho>> "Curso". Já o conceito "Administrador" não possui um conceito que seja subordinado a ele em um nível inferior.

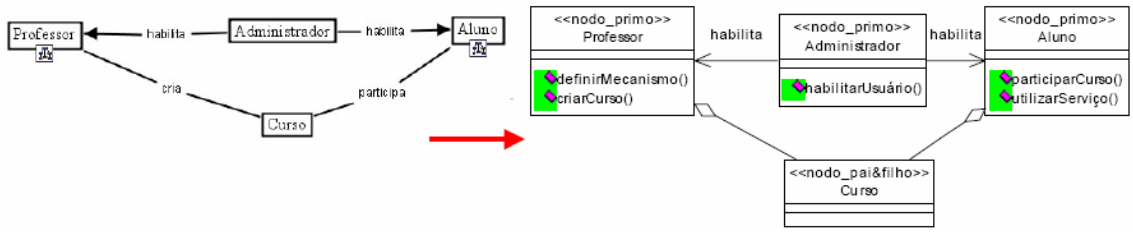


Figura 24 - Utilização do estereótipo <<nodo_primo>>

2.9.5 O estereótipo <<nodo_composicao>>

Intenção: ocorrer quando um <<nodo_pai&filho>>, um <<nodo_primo>> ou um <<nodo_filho>> informa algo sobre o conceito de nível superior, que pode ser um <<nodo_pai>>, um <<nodo_pai&filho>> ou um <<nodo_primo>>.

Problema: informar algo sobre o nível superior.

Solução: criar um relacionamento de agregação.

Participantes e colaboradores: <<nodo_agregacao>> e <<nodo_pai&filho>>, <<nodo_primo>>, e <<nodo_filho>>. Já no nível superior, <<nodo_pai>>, <<nodo_pai&filho>> ou <<nodo_primo>>.

Implementação: é o tipo de relacionamento de agregação (ou "contém-um"), cuja inclusão significa que o objeto constitui uma parte do objeto que o contém. Assim, o exemplo abaixo mostra que "Curso" tem "Serviços", ou seja, o "Curso" é feito de

"Serviços" e outras partes. Portanto, esse tipo de relacionamento "contém-um", denominado composição, é indicado pelo losango cheio.

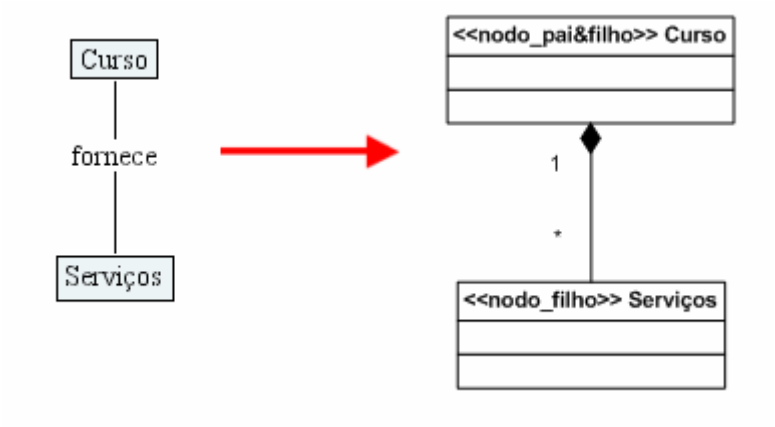


Figura 25 - Utilização do estereótipo <<nodo_composicao>>

2.9.6 O estereótipo <<nodo_agregacao>>

Intenção: o <<nodo_agregacao>> também ocorre quando um <<nodo_pai&filho>>, um <<nodo_primo>> ou um <<nodo_filho>> informa algo sobre o conceito de nível superior, que pode ser um <<nodo_pai>>, um <<nodo_pai&filho>> ou um <<nodo_primo>>.

Problema: informar algo sobre o nível superior.

Solução: relacionamento de agregação. Porém as informações não ficam armazenadas no conceito mais geral, assim, nesse tipo de relacionamento, também seria possível utilizar o relacionamento de associação binária, mas usamos o <<nodo_agregacao>> para identificar qual conceito é hierarquicamente superior a outro.

Participantes e colaboradores: <<nodo_agregacao>> e <<nodo_pai&filho>>, <<nodo_primo>>, e <<nodo_filho>>. Já no nível superior <<nodo_pai>>, <<nodo_pai&filho>> ou <<nodo_primo>>.

Implementação: existem realmente dois tipos diferentes de relacionamento de agregação (ou "tem-um"). Isso quer dizer que um objeto pode conter um outro objeto, sendo que esse objeto nele contido pode ser uma parte do objeto contêiner ou não. Dessa maneira, na Figura 26 a classe "Curso" contém "Mecanismos", sendo que este não faz parte de "Curso", mas ainda assim podemos dizer que ele a contém.

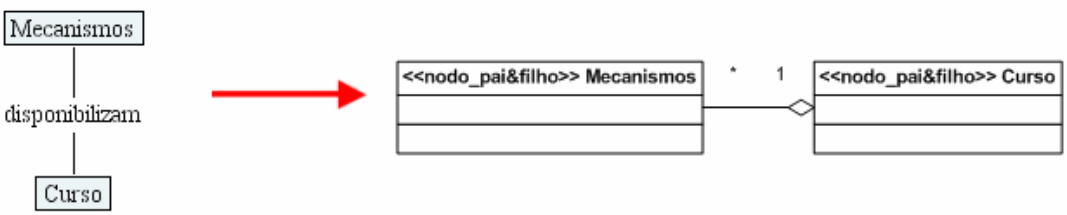


Figura 26 - Utilização do estereótipo <<nodo_agregacao>>

3 Trabalhos relacionados

Nesta seção, trabalhos relacionados são apresentados, e, para tanto, será feita a análise de algumas soluções para elicitação e gerenciamento de requisitos em projetos XP. No entanto, algumas ferramentas oferecem mais funcionalidades além da elicitação de requisitos usando *User Stories*, como por exemplo, gerenciamento de tarefas, testes de aceitação e trabalho cooperativo suportado por computador (CSCW) [10].

Em outras palavras, alguns trabalhos suportam apenas o gerenciamento de *User Stories*, como na subseção 3.3 por exemplo; enquanto outros apresentam uma abordagem mais holística, como na subseção 3.2 por exemplo.

3.1 PAM

É um experimento feito através de um protótipo, e baseado no CSCW, o qual pressupõe que o suporte de CSCW para XP é benéfico para equipes XP dispersas e que provê suporte à fase de planejamento de XP. Assim, na Figura 27 podemos ver o desmembramento de um projeto na ferramenta PAM [WIL 2003].

Além disso, PAM é uma aplicação *desktop*, ou seja, uma tecnologia que oferece um suporte mais sofisticado ao projeto de interfaces para computadores e tecnologias de rede, o que significa que é uma tecnologia mais "madura". O criador da aplicação PAM deixa claro que a tecnologia *web* possui suas vantagens, como por exemplo, uma facilidade maior em termos de acessibilidade (com a utilização de *browsers*) e, assim, uma versão *web* de PAM pode ser criada futuramente.

Figura 27 – Desmembramento de um projeto na ferramenta PAM

[10] CSCW é a abreviatura de "Computer Supported Cooperative Work", traduzido em Português como Trabalho Cooperativo Suportado por Computador (http://pt.wikipedia.org/wiki/CSCW).

Além do mais, essa interface organiza de forma hierárquica os requisitos funcionais de grupos de usuários em três atividades, as quais são:

- Produção – criação e persistência de *User Stories*, *releases* e cronogramas de iterações. Além disso, há suporte para criação e testes de código;
- Comunicação – suporte para *chats* para pares, grupos e interno/externo ao projeto;
- Coordenação – controle de sessões com usuários que podem realizar atualizações em tempo real no projeto.

É importante ainda vermos, na Figura 28, a comparação de um típico cartão *User Story* com um outro cartão criado pelo sistema PAM.

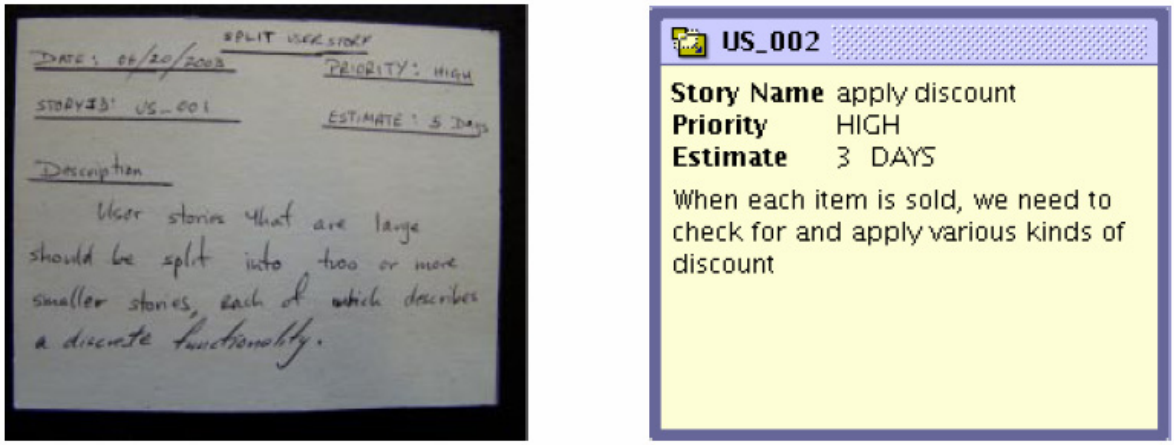


Figura 28 - Comparação de um típico cartão de *User Story* com um cartão criado pelo sistema PAM

Assim, a pesquisa conclui que PAM atende a reais necessidades de *Extreme Programming*, tais como manutenção de *software* inadequada e falta de escalabilidade. Porém, ela continua sendo melhorada em trabalhos futuros.

Críticas: segundo uma avaliação dos usuários, algumas desvantagens relativas ao uso da ferramenta PAM foram apresentadas. Por exemplo, uma comunicação face-a-face com o usuário é mais eficiente do que uma comunicação remota. No entanto, avaliações podem ser realizadas para analisar essas desvantagens, como comparar equipes XP dispersas que utilizam PAM com equipes XP dispersas que utilizam os métodos tradicionais.

Foi realizada uma avaliação formal que destacou os benefícios da ferramenta em equipes XP, assim como mostrou o que foi feito para melhorar a utilidade e capacidade da ferramenta.

A avaliação da ferramenta PAM ainda é prematura e novos testes devem ser realizados, sendo que o resultado destes testes poderá ser utilizado para otimizar PAM para um futuro uso comercial [WIL 2003].

3.2 XPSwiki

XPSwiki é uma ferramenta *web* desenvolvida para suportar o Jogo de Planejamento, a qual é baseada na tecnologia Wiki[11], sendo que a Figura 29 mostra uma captura de tela da interface gráfica da ferramenta. Além disso, ela acompanha as mudanças do projeto através de um sistema de versões, assim como também gerencia a persistência de dados para *releases*, iterações, *User Stories*, tarefas e testes de aceitação, e oferece uma estrutura hierárquica do planejamento e artefatos do projeto [PIN 2003] [WIL 2003].

São destacadas as seguintes vantagens da ferramenta [ANG 2006]:

- Usuários e desenvolvedores podem manipular os dados diretamente da *web* sem a necessidade de utilizar documentos em papel ou intalar aplicações proprietárias;

- Equipes dispersas podem cooperar e ter uma visão completa do projeto;

- *User Stories*, tarefas e outras informações relevantes ao projeto são disponibilizadas diretamente na *web*.

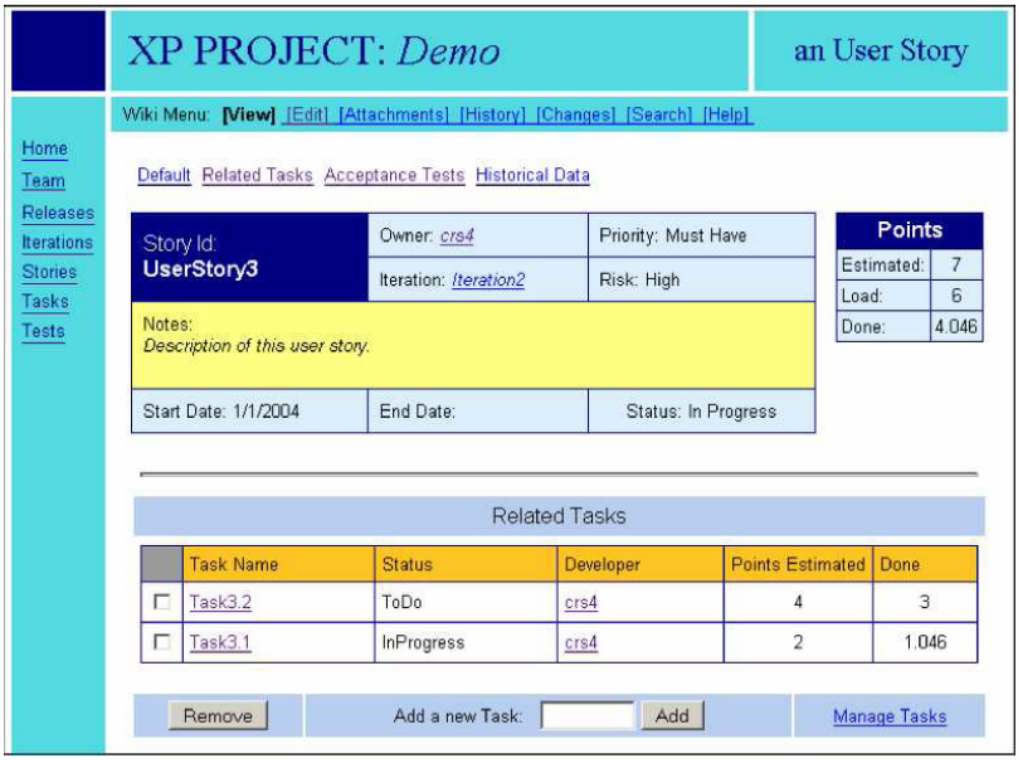


Figura 29 - Página *web* da XPSwiki [ANG 2006]

Crítica: XPSwiki suporta uma abordagem holística para suporte à XP, entretanto, ela é limitada em alguns pontos:

- Falta de um mecanismo adequado para colaboração síncrona;

[11] O termo Wiki é utilizado para identificar um tipo específico de coleção de documentos em hipertexto ou o software colaborativo que permite a edição coletiva dos documentos usando um singelo sistema e sem que o conteúdo tenha que ser revisto antes da sua publicação (http://pt.wikipedia.org/wiki/Wiki).

- Falta de um mecanismo de percepção, o que é conceituado como a contextualização das atividades individuais através da compreensão das atividades realizadas por outras pessoas. Em outras palavras, refere-se a ter conhecimento das atividades do grupo, saber o que aconteceu, o que está acontecendo e/ou o que poderá vir a acontecer, além do próprio conhecimento do que é esse trabalho e o grupo [PIN 2006];
- Não oferece mecanismos para integração e teste unitário;
- A tecnologia Wiki usada é suscetível a problemas de resolução e conflitos de atualização.

É importante ainda pontuar que um importante *feedback* foi recebido dos usuários de XPSwiki, o qual sugeriu que uma ferramenta *web* de suporte à XP é importante para promover essa ferramenta no mercado, bem como ajudar os desenvolvedores no aprendizado do Jogo de Planejamento.

3.3 DotStories

DotStories é uma ferramenta *web* desenvolvida para substituir os cartões *User Story* manuais, que suporta múltiplos projetos, os quais são reunidos como uma coleção de várias páginas *web* que definem um grupo de *User Stories*, apresentadas como "*User Story Teglets*", sendo que uma pequena área retangular na página *web* facilita a manipulação, a visualização de cartões *User Stories* e a formatação, tais como identação, negrito e assim por diante. Assim, podemos ver um exemplo de "*User Story Teglets*" na Figura 30, e observar que é permitida a criação, edição, divisão e exclusão de *User Stories*, e ainda que *Teglets* podem ser também agrupadas. Isso significa que DotStories suporta projetos com várias centenas de *User Stories* [REE 2002] [WIL 2003].

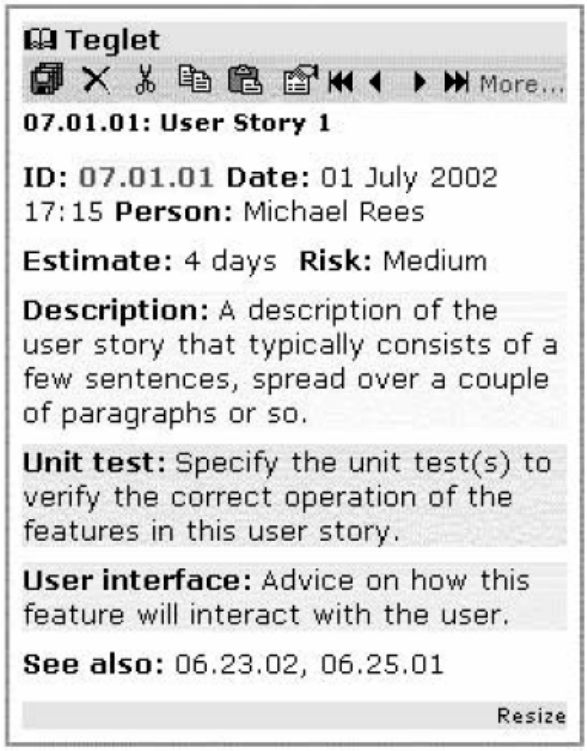


Figura 30 - Exemplo de *User Story Teglet* da ferramenta DotStories [REE 2002]

Por sua vez, a janela *Teglet* possui as seguintes características:

- A barra de título indica o tipo de *Teglet* e permite que ela seja movida ao redor da página, assim como incluída em um grupo de *User Stories*;
- A barra de ferramentas possui opções para edição e formação de texto;
- A área principal de edição oferece uma forma DHTML;
- A barra "*Resize*", na parte inferior, permite mudar o tamanho da tela *Teglet*.

Assim, na Figura 31 podemos ver a interface principal de gerenciamento de *User Story Teglet*, que permite a inclusão de novas páginas, a remoção de páginas já existentes, e a definição da ordem da página, sendo que algumas funcionalidades estão disponíveis ao clicar no botão direito do *mouse*, tais como: criação de novas *Teglets*, funcionalidades de copiar e colar *Teglets* individuais ou grupo de *Teglets*, e salvar *Teglets*.

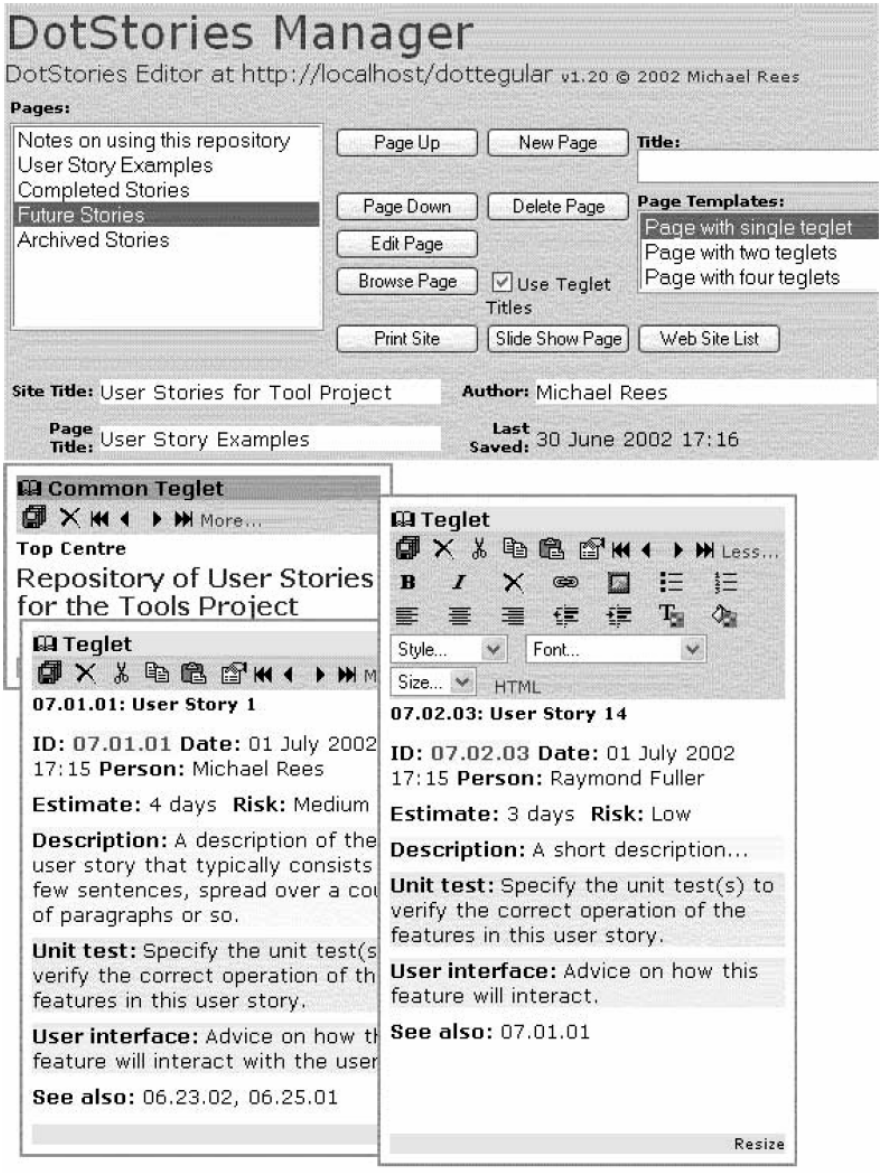


Figura 31 - Exemplo de gerenciamento de *User Story Teglet* [REE 2002]

Crítica: DotStories é uma alternativa ao uso de cartões *User Stories* normais, entretanto, embora Rees afirme que *User Story* é o principal trabalho produzido pela XP para dirigir o processo, essa ferramenta não oferece um bom suporte à colaboração de informações, não atendendo às reais necessidades para times dispersos [WIL 2003].

3.4 MILOS

É uma ferramenta *web* para suporte à XP distribuída, que propõe solução para suporte a equipes de *software* virtual com comunicação, colaboração e coordenação, a qual foi originalmente desenvolvida para suportar execução de processos e aprendizado organizacional para equipes virtuais. MILOS inclui ainda características como

mecanismo de *Workflow*[12] para gerenciamento de tarefas; *NetMeeting* para suporte à Jogo de Planejamento e programação em pares; uma lista de tarefas pessoal; ajuda eletrônica; e autenticação de usuários. Além disso, permite inserir, excluir e editar *User Stories* e tarefas [MAU 2002] [WIL 2003].

Após a criação de um projeto, o usuário pode incluir as *User Stories*, conforme mostra a Figura 32.

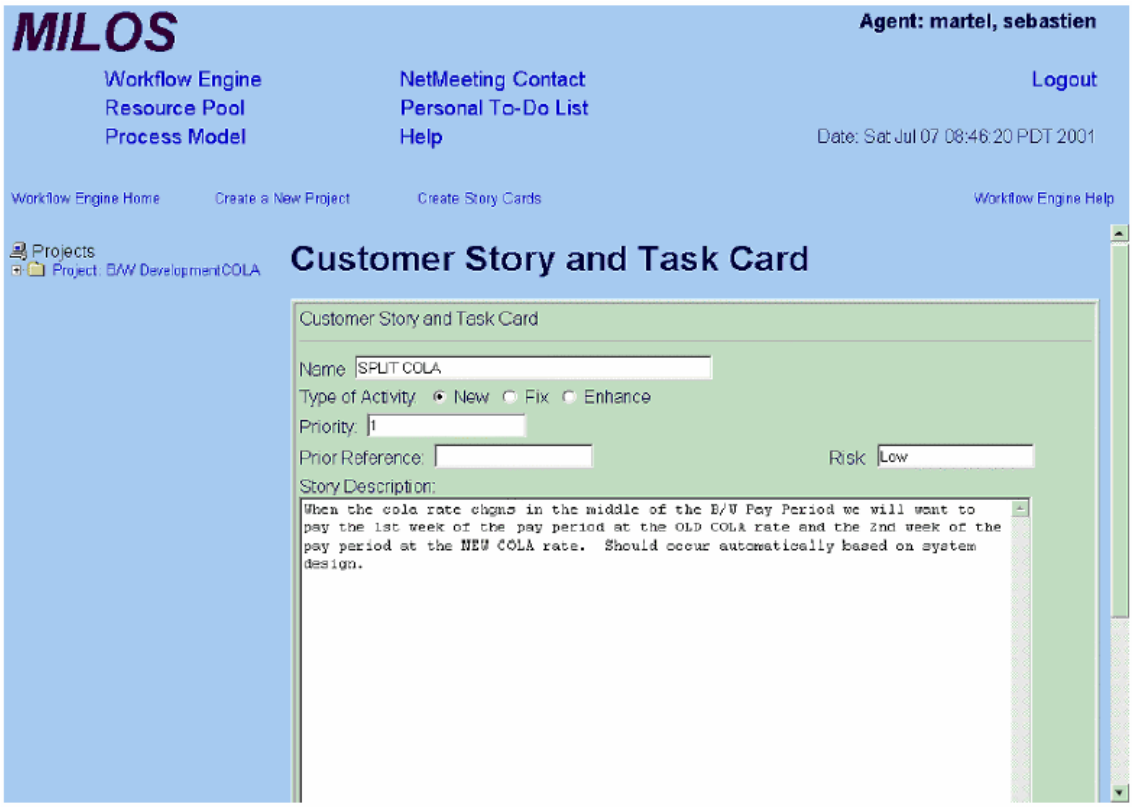


Figura 32 - Exemplo de uma *User Story* na ferramenta MILOS [MAU 2002]

Sendo assim, para cada *User Story* criada, uma tarefa é gerada automaticamente, sendo que, através da interface da Figura 33, o usuário pode decompor a tarefa em subtarefas e, em seguida, descrever a tarefa em maiores detalhes.

[12] Workflow pode ser definido como seqüência de passos necessários para que se possa atingir a automação de processos de negócio, de acordo com um conjunto de regras definidas (http://pt.wikipedia.org/wiki/Workflow).

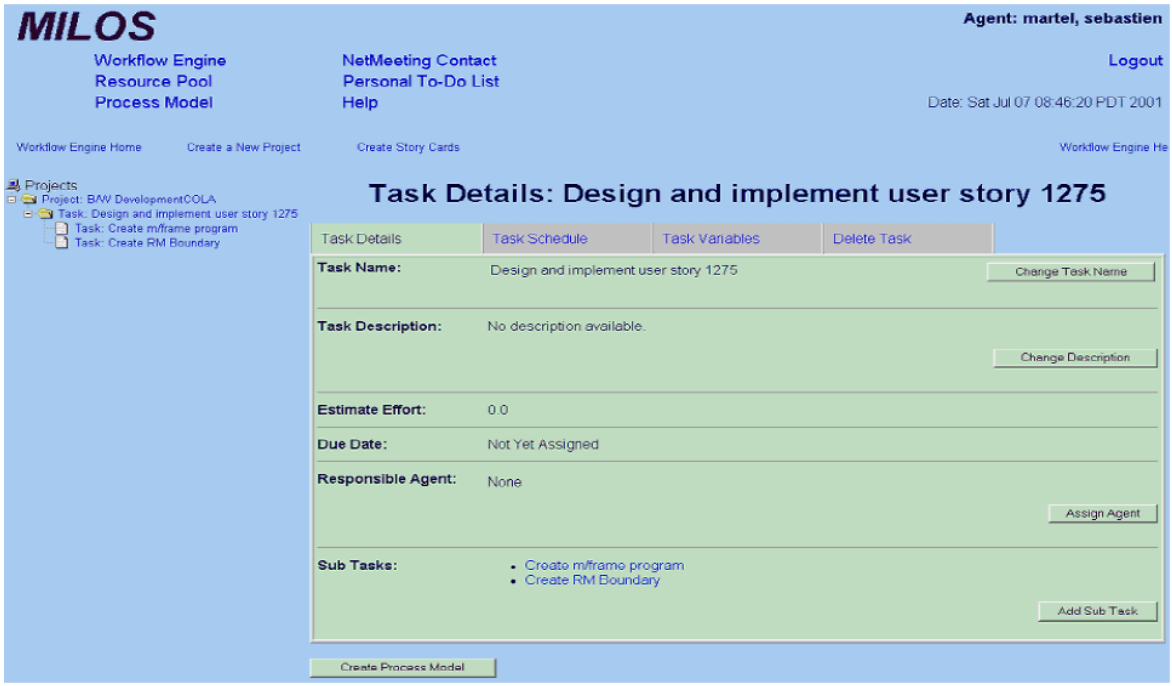


Figura 33 - Organização de tarefas na ferramenta MILOS [MAU 2002]

Crítica: MILOS é parcialmente distribuída, e oferece mais funcionalidades do que DotStories (seção 3.3), entretanto, também apresenta limitações, as quais são:

- O uso de *NetMeeting* restringe o usuário a utilizar a plataforma Microsoft Windows;
- Como DotStories, MILOS suporta manipulação direta a *User Stories* e tarefas, porém não é uma manipulação colaborativa;
- O uso de vídeo conferência é destacado nesta ferramenta, porém isso é freqüentemente desnecessário, pois, geralmente os desenvolvedores já se conhecem.

3.5 TUKAN

TUKAN é uma ferramenta que suporta programação distribuída e Jogo de Planejamento, e que oferece reuniões virtuais, colaboração síncrona, canais de comunicação, e fornecimento de contexto aos membros do grupo, assim como também várias técnicas de visualização são aplicadas para suportar programação em pares e integração, e em que *User Stories* e anotações são permitidas, e cuja comunicação é feita via áudio e *chat* [SCH 2001].

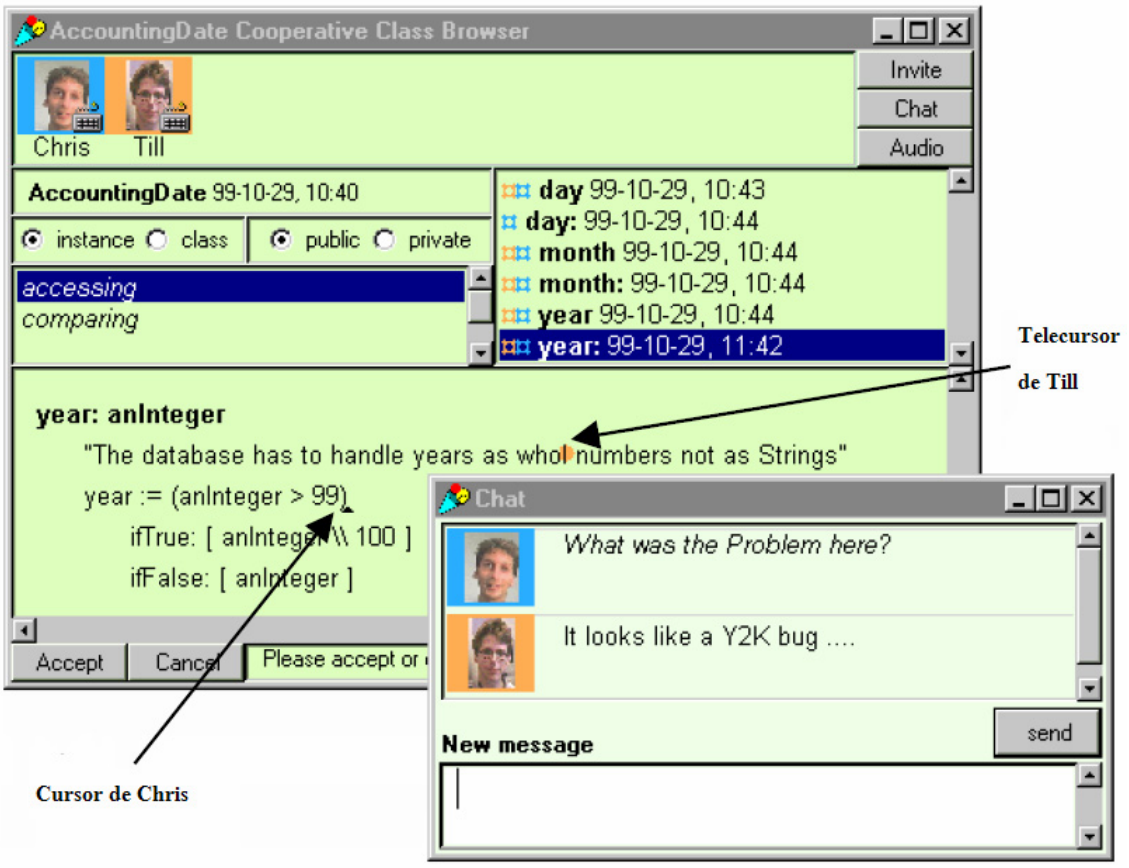


Figura 34 - Cooperação de uma classe e a utilização de *chat* na ferramenta TUKAN

Como é apresentado na Figura 34, na parte superior direita da janela, uma lista hierárquica de métodos é disponibilizada, já o código dos métodos são exibidos na parte inferior da janela, o que cria o foco do usuário. Por sua vez, na perspectiva do usuário, navegar através do código é como caminhar através de um espaço de *software* virtual, sendo que seu foco está sobre o método exibido no momento.

Além disso, a presença de outros usuários é refletida na lista de métodos nas figuras coloridas ao lado de cada método, em que cores e figuras são utilizadas para indicar possíveis conflitos entre os usuários.

Finalmente, quando uma colaboração de código síncrona é realizada, a presença de um outro desenvolvedor pode ser percebida na parte superior da ferramenta, assim, se os desenvolvedores estão localizados de forma distribuída, eles podem utilizar a ferramenta de *chat* ou áudio para conversação.

Crítica: a TUKAN oferece uma abordagem holística para trabalhar com XP em ambientes distribuídos, oferece uma melhor comunicação e colaboração do que *NetMeeting*, devido ao uso de replicação (oposto à abordagem utilizada por *NetMeeting*). Entretanto, não encontramos informações sobre a manipulação direta de cartões *User Story* com a ferramenta TUKAN.

3.6 Web-Desktop

Web-Desktop foi uma das primeiras abordagens na tentativa de prover um suporte a equipes XP em ambientes distribuídos, sendo que seus autores afirmam que somente quatro das práticas de XP são afetadas em um ambiente distribuído: Jogo de Planejamento, programação em pares, integração contínua e cliente *on-site*, e cujo objetivo original desta pesquisa foi proporcionar um ambiente distribuído de trabalho sem impactar o ambiente XP de forma negativa [KIR 2001]. Além disso, a comunicação pode ser suportada através do uso de tecnologias tais como vídeo conferência, *e-mail* e *chat*.

Web-Desktop utiliza ainda um modelo de arquitetura cliente/servidor, sendo que aplicações para suporte aos processos de desenvolvimento e gerenciamento ficam disponíveis para *download* em servidores dedicados. Por sua vez, as atividades XP como Jogo de Planejamento, programação em pares, integração contínua e cliente *on-site* são suportadas através de ferramentas já existentes, tais como: vídeo conferência, *NetMeeting*, *e-mail* e aplicações de compartilhamento de informações.

Portanto, os maiores desafios encontrados por Web-Desktop foram:

- A captura de *User Stories* é feita através de arquivo texto;
- Problemas de interoperabilidade com o sistema operacional;
- Redução de capacidade da rede por causa das conferências;
- Falta de um acesso uniforme ao repositório de código fonte;
- Questões de internacionalização tais como *layouts* de teclados e caracteres de diferentes países.

Críticas: as limitações dessa abordagem estão relacionadas principalmente a questões de interoperabilidade, uma vez que aplicações diferentes foram utilizadas para o mesmo propósito, dificultando modificações em novos ambientes. Além disso, ela demonstra que existem ferramentas que podem ser usadas para suportar XP em ambientes distribuídos, entretanto, considerações especiais podem ser feitas em termos de interoperabilidade, aplicabilidade e flexibilidade sobre as ferramentas selecionadas. Portanto, os resultados demonstraram que ferramentas customizáveis oferecem um grande potencial para suportar XP em ambientes distribuídos.

3.7 Storymanager

Storymanager é uma solução para gerenciamento de *User Stories* e tarefas, cuja proposta foi integrada ao ambiente Eclipse, como podemos ver na janela principal da ferramenta na Figura 35. por sua vez, Eclipse é um ambiente de desenvolvimento de *software* e um *framework* de integração apropriado para o desenvolvimento XP, sendo que um *framework* possibilita um canal de comunicação para a equipe durante todo o ciclo de vida do desenvolvimento, assim, por exemplo, durante o Jogo de Planejamento, a equipe trabalhou com um *plug-in* Storymanager [KAA 2004].

Além disso, o Storymanager permite uma especificação de atributos de *User Stories* e tarefas de acordo com as necessidades do projeto, sendo que o programa possibilita completar cartões *User Story* e de tarefas de acordo com atributos específicos do projeto, uma vez que há uma funcionalidade de geração automática de um identificador único para *User Stories* e tarefas. Certos atributos, entretanto, são obrigatórios, como situação, descrição, identificador de *release*, e identificador de iteração, assim como outros atributos são definidos pelos usuários e são opcionais.

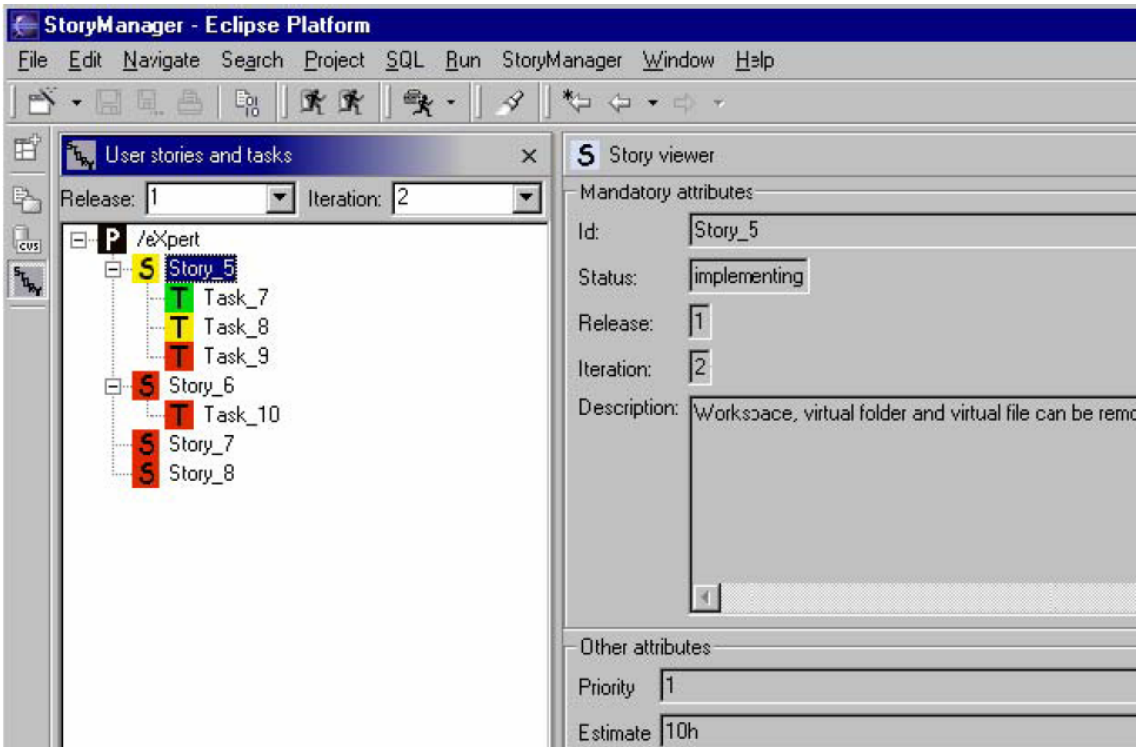


Figura 35 - A janela principal de Storymanager

Críticas: um estudo foi feito por Kääriäinen [KAA 2004], sendo que a ferramenta Storymanager foi usada em um estudo de caso onde uma aplicação móvel para mercados reais foi criada. No entanto, a ferramenta foi abandonada pela equipe somente depois de duas *releases*, e a razão da provável falha foi apresentada. Portanto, os principais resultados apresentados foram a dificuldade de utilização da ferramenta e a falta de poderosos recursos visuais para visualização de *User Stories*. Ou seja, o suporte à XP através de uma ferramenta deve oferecer o que o método tradicional já provê, por exemplo, agilidade, além de agregar algum valor, o que não foi o caso da Storymanager.

4 RequirementX

Este capítulo apresenta o RequirementX, que constitui uma proposta para a formalização de requisitos de *software* criado por usuários com a metodologia *Extreme Programming*. RequirementX é uma extensão do software CmapTools que foi especializada para alcançar o objetivo proposto por este trabalho.

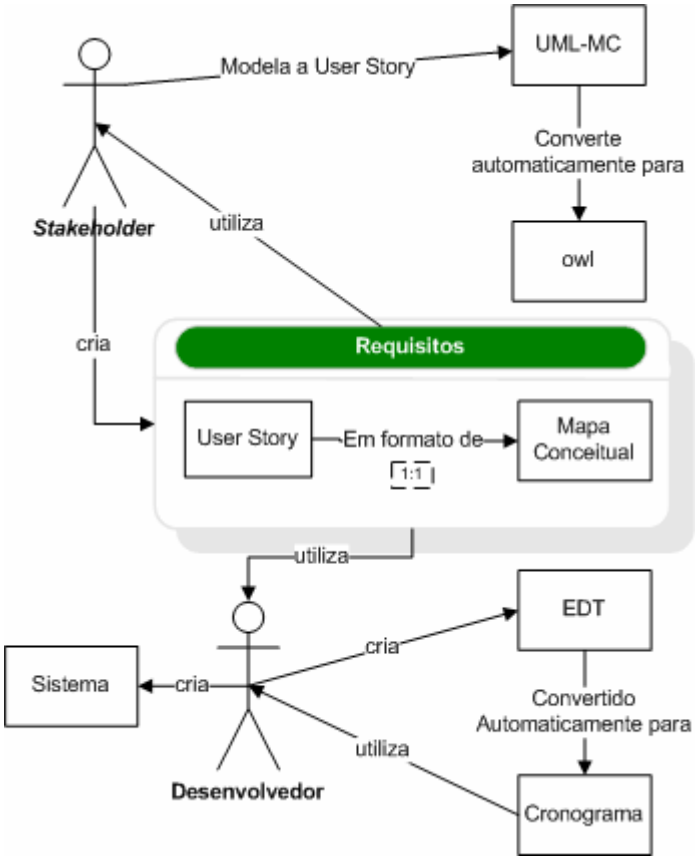


Figura 36 - Apresentação da Proposta RequirementX

A Figura 36 apresenta de forma resumida a proposta da ferramenta RequirementX. O s*takeholder* cria as *User Stories* em formato de Mapa Conceitual, sendo que deve existir uma *User Story* para um Mapa Conceitual. A decisão de criar esta relação "1:1" surgiu após alguns testes durante a pesquisa, verificou-se que um Mapa Conceitual representando mais de uma *User Story* é mais difícil de ser manipulado e gerenciado.

O *stakeholder* poderá adicionar uma semântica ao requisitos modelando as *User Stories* através da notação UML-MC. Esta modelagem é realizada manualmente pelo *stakeholder* através de funcionalidades da ferramenta RequirementX. Após formalizada, esta *User Story* poderá ser exportada automaticamente para arquivos OWL, provendo assim interoperabilidade com outras aplicações.

O desenvolvedor poderá utilizar as *User Stories* para criar as Estrutura de Desmembramento de Trabalho (EDTs). Estes EDTs devem ser criados manualmente pelo desenvolvedor, através da aplicação RequirementX, e podem ser convertidos automaticamente para um cronograma. Este cronograma poderá ser exportado para

arquivos XML compatíveis com o *MS Project*. Com os cronogramas criados, os desenvolvedores podem iniciar o desenvolvimento do sistema.

4.1 Arquitetura do RequirementX

O RequirementX possui cinco módulos básicos, conforme podemos ver na Figura 37.

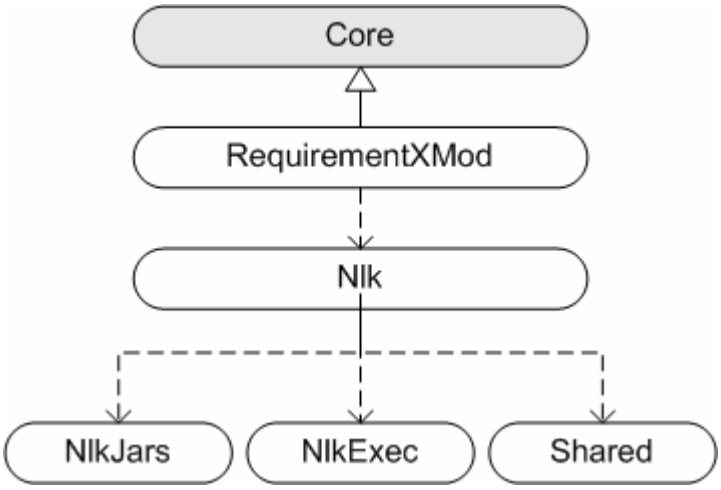


Figura 37- Módulos da Arquitetura RequirementX

A seguir, descreveremos cada um dos módulos da arquitetura RequirementX:

- *Core* - é o módulo principal do *CmapTools* (*Kernel*), e é nele que todos os outros módulos são baseados;
- *Nlk* - esse módulo possui os novos módulos do *CmapTools*, ou seja, quando um novo módulo é criado, é necessário criar um diretório dentro do diretório "nlk". Por exemplo, o diretório "RequirementX" foi criado para a nossa aplicação, e dentro desse diretório há várias classes (Figura 38), sendo que a principal é a "RequirementXMod", que representa o novo módulo e herda as funcionalidades do módulo "*Core*";
- *NlkJars* - esse módulo possui todas as bibliotecas de terceiros, ou seja, que não foram construídas pela equipe desenvolvedora do *CmapTools*;
- *NlkExec* - esse módulo possui os arquivos de lote para compilação, preparação de arquivo JAR e execução dos módulos;
- *Shared* - esse módulo possui alguns arquivos de gerência de arquivos fontes, realizada pelo CVS[13].

Passamos a analisar o módulo RequirementX em maiores detalhes no diagrama de classes da Figura 38.

[13] http://www.cvshome.org

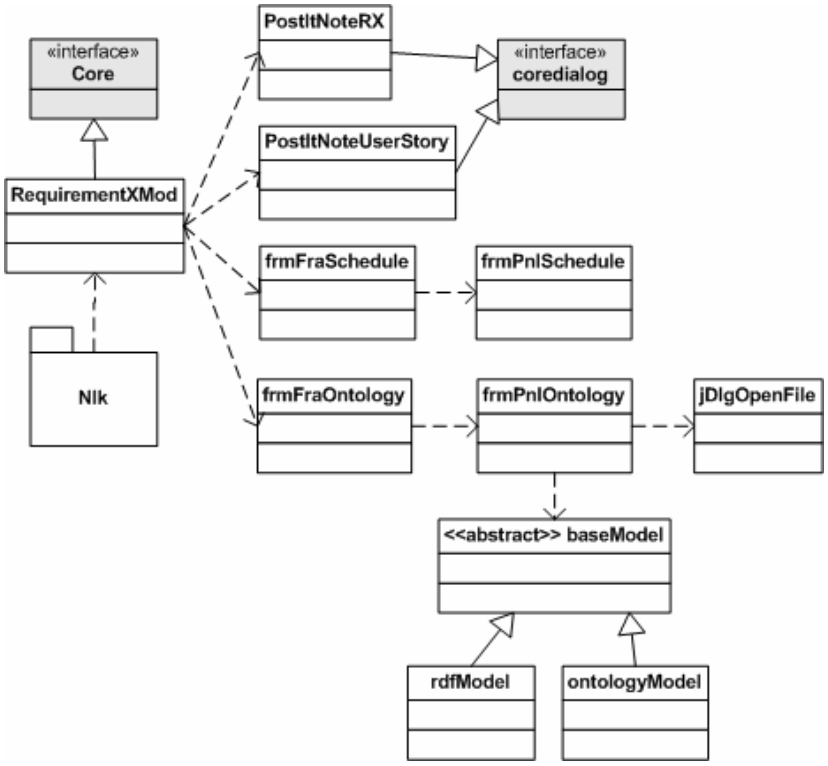


Figura 38 - Diagrama de classes da ferramenta RequirementX

Agora, vamos descrever alguns detalhes das classes da arquitetura da ferramenta RequirementX, que são:

- *RequirementXMod* – é a classe principal do módulo RequirementX, e herda as funcionalidades do módulo central "*Core*";
- *PostItNodeRX* e *PostItNoteUserStory* – a classe *postItNodeRX* representa uma interface que capta informações relativas ao projeto e a classe *PostItNoteUserStory* representa uma interface que capta informações relativas as *User Stories*, conforme mostra a Figura 38, sendo que ambas herdam as funcionalidades do módulo "*coredialog*". Além disso, essas interfaces são do tipo *Post-it*[14], ou seja, são interfaces pequenas que podem ser acessadas de forma rápida e fácil através do botão direito do *mouse*;
- *frmFraSchedule* e *frmPnlSchedule* – representam a interface que converte a EDT criada pelo desenvolvedor em um cronograma;
- *frmFraOntology* e *frmPnlOntology* – representam a interface que permite ao usuário atribuir uma semântica ao Mapa Conceitual criado pelo usuário (que representa uma *User Story*), utilizando a notação UML-MC e posteriormente exportá-lo para um arquivo OWL. Além disso, é a interface que provê interoperabilidade com outros sistemas;
- *jDlgOpenFile* – é utilizada para visualizar os arquivos OWL exportados;

[14] O Post-it é um pequeno papel (de 7,5 cm de área) com um adesivo de fácil remoção atrás de si, de forma que seja facilmente pregado e arrancado de uma superfície. Usado para deixar lembretes (http://pt.wikipedia.org/wiki/Post-it).

- *baseModel* – é uma interface abstrata que possui funcionalidades comuns às classes *rdfModel* e *ontologyModel*;
- *rdfModel* – classe utilizada para criação e manipulação de arquivos RDF;
- *ontologyModel* – classe utilizada para criação e manipulação de ontologias como OWL e DAML.

Explicaremos no tópico a seguir como o módulo RequirementX foi desenvolvido.

4.2 Criação do módulo RequirementX

Para a programação do novo módulo utilizamos o *software* NetBeans IDE 5.0[15], um ambiente de desenvolvimento que permite escrever, compilar, depurar e instalar programas na linguagem. No entanto, o NetBeans será utilizado apenas para a programação, pois o processo de compilação e execução será realizado de acordo com as convenções do *CmapTools*, que, por sua vez, será explicado a seguir. É importante também citar que o NetBeans IDE é um produto livre, sem restrições de como ele pode ser usado, e cuja utilização é demonstrada abaixo.

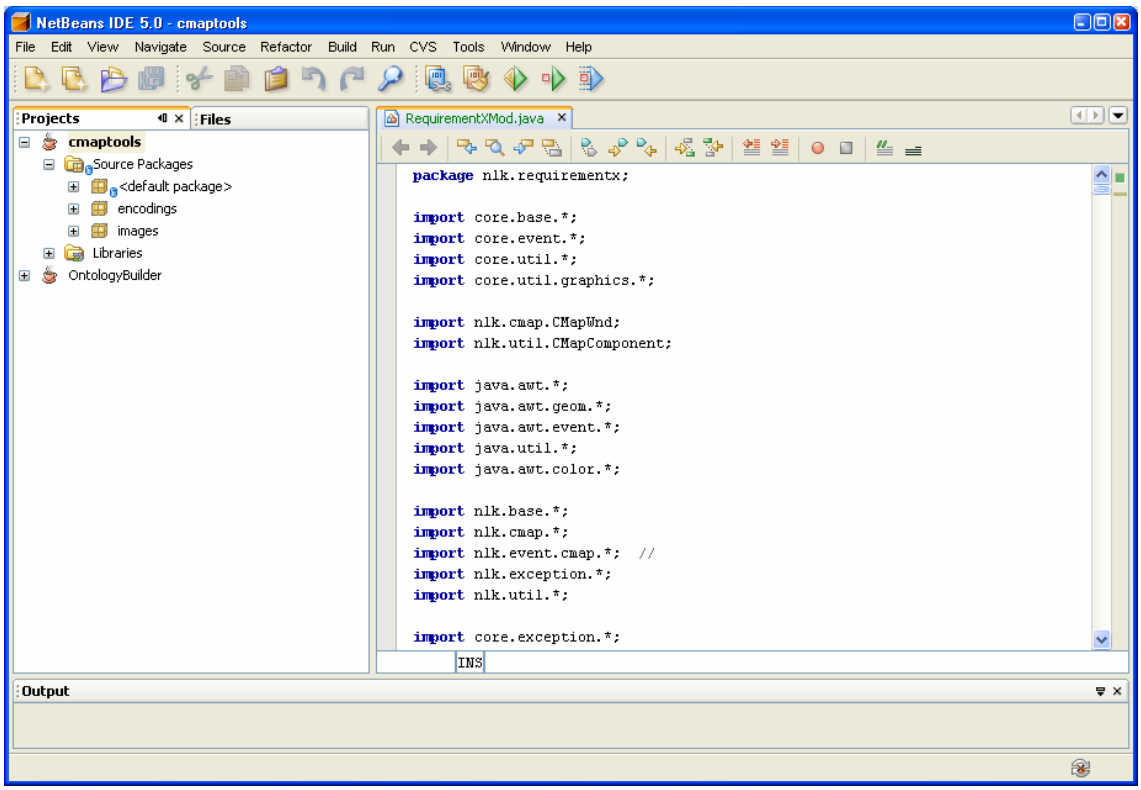


Figura 39 - Utilização do *software* NetBeans 5.0 para a criação do novo módulo

[15] http://www.netbeans.org

Entretanto, antes de demonstrar como criar o novo módulo, é importante deixar claro alguns conceitos sobre o *CmapTools* para melhor explicar os passos a seguir, o qual possui dois tipos principais de telas, que são descritas a seguir:

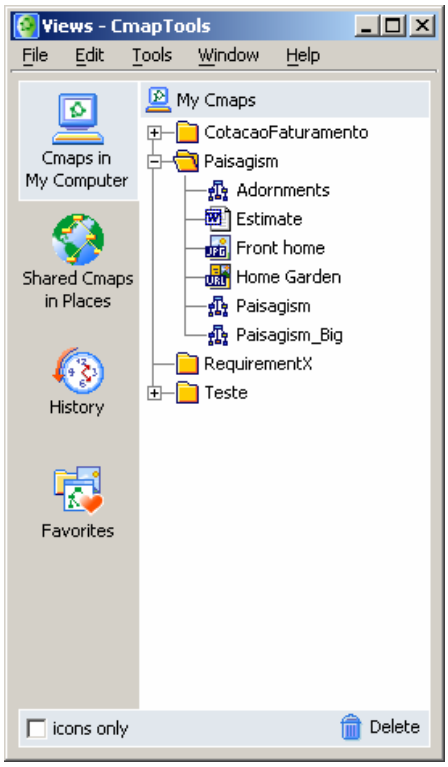


Figura 40 - A janela "*Views*"

a) A janela "*Views*", que é a tela inicial do *CmapTools*, e utilizada como organizador central da ferramenta. Assim, é possível organizar os seus Mapas Conceituais, chamados Cmaps, e recursos em pastas, no disco rígido do próprio computador e sobre servidores remotos que são compartilhados com uma comunidade de Cmap.

b) A janela "*Window*", onde será criado o Mapa Conceitual.

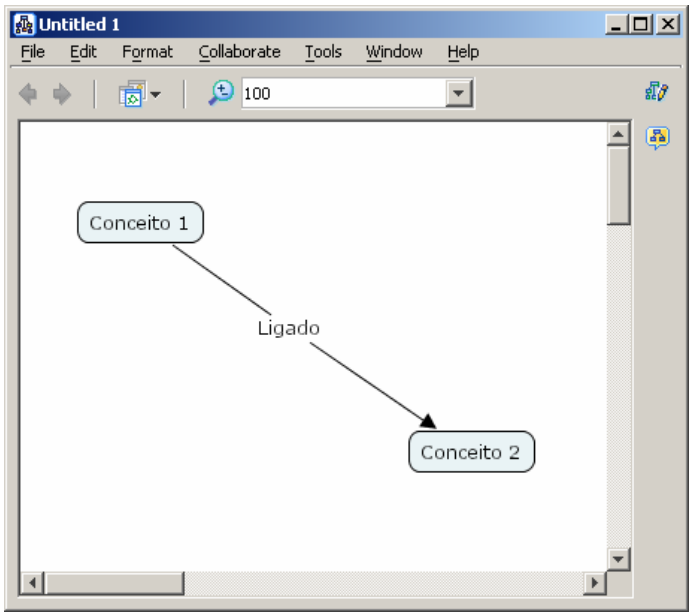


Figura 41 - Janela "*Window*"

Dessa maneira, o novo módulo criado herdará todas as funcionalidades básicas do "*Core*" *CmapTools*, mas permitirá criar especializações, assim, será possível criar, por exemplo, novos itens de menu na janela "*Views*" ou na janela "*Window*" que executam novas funcionalidades.

4.3 Especializações criadas no novo módulo para o RequirementX

Algumas especializações foram criadas na ferramenta RequirementX para facilitar o trabalho dos desenvolvedores na formalização do Mapa Conceitual, ou seja, na captação de informações necessárias para posteriormente criar um cronograma, como podemos ver a seguir:

a) Opções para mudar automaticamente o tipo, a cor e o formato dos conceitos

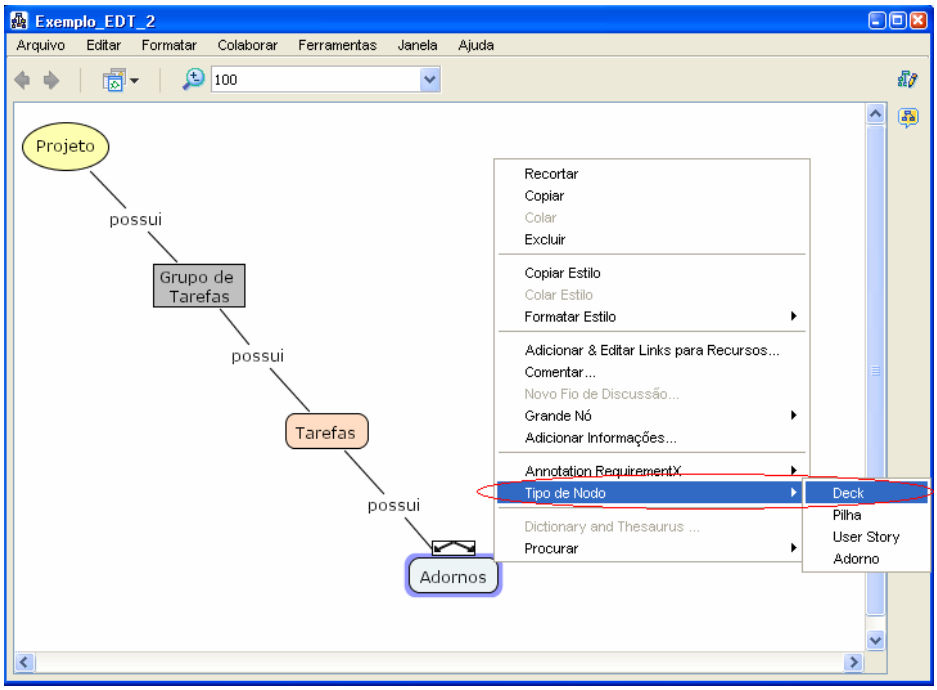


Figura 42 - Opções para mudar automaticamente o tipo, a cor e o formato dos conceitos

b) Interface para captar informações necessárias no cronograma

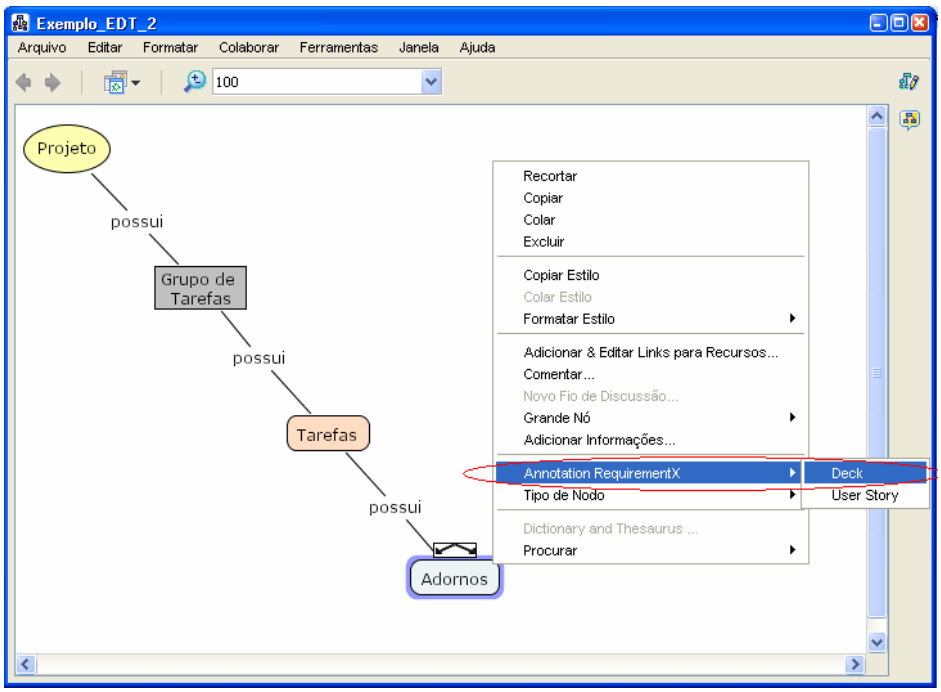


Figura 43 - Opção para captar informações necessárias no cronograma

Além do mais, foram definidos tipos de nodos para desmembrar um projeto em seus componentes e partes, auxiliando assim na identificação das tarefas, os quais são:

- *Deck* – é o projeto, ou seja, a compreensão atual do cliente daquilo que deve ser realizado para criar o sistema desejado, o qual define o domínio do conhecimento ao qual o mapa se refere. Além disso, esse termo possui as seguintes propriedades:
  - Data inicial do projeto (tipo data);
  - Data final do projeto (tipo data);
  - Descrição (texto com 450 caracteres);
  - Cor (1=amarelo);
  - Formato (O=Oval).
- Pilha - é a tarefa de resumo, que possui as seguintes características:
  - Descrição (texto com 300 caracteres);
  - Cor (3=cinza);
  - Formato (R=Retangular).
- *User Story* – é uma história de usuário que será transformada em uma tarefa, e que tem as seguintes propriedades:
  - ID (código seqüencial da *User Story*);
  - *Task Dependency* (valores possíveis: *Finish-to-Start(FS); Start-to-Start(SS); Finish-to-Finish(FF); Start-to-Finish(SF)*);
  - Duração (é a duração em dias do tipo inteiro longo);
  - Predecessores (do tipo inteiro longo);
  - Recursos (texto com 450 caracteres);
  - Percentual de utilização do recurso (do tipo *float*);
  - Prioridade (do tipo inteiro longo);
  - Cor (2= vermelho);
  - Formato (RA=Retângulo de cantos arredondados).
- Adorno - são *User Stories* que acrescentam detalhes a outras *User Stories*, pilhas ou *Decks* que são usados para eliminar a complexidade de uma *User Story*. Além disso, um adorno ligado ao *Deck* pode ser chamado de cartão de visão (*Vision Card*), pois faz referência a todo o projeto. Assim, podemos dizer que o adorno é um MC simples, utilizado apenas para descrever um conceito, e que tem as seguintes características:
  - Descrição (texto com 300 caracteres);
  - Cor (4=azul);
  - Formato (RA=Retângulo de cantos arredondados).

Como exemplo, temos a tela que capta informações referentes a um conceito do tipo *User Story* (pequena tela no centro da Figura 44), e que será transformado em uma tarefa.

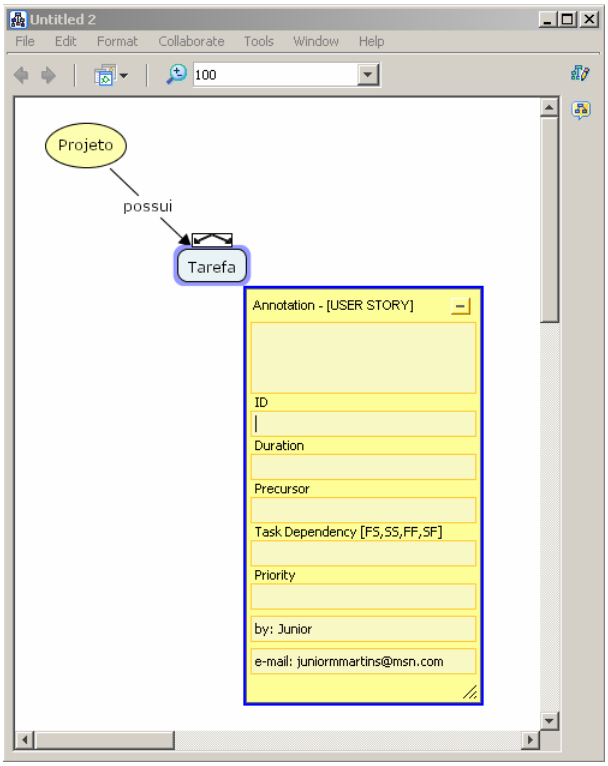


Figura 44 - Interface para captar informações de conceitos do tipo User Story

Além disso, o nodo do tipo *Deck* também possui uma interface específica para a captação de informações, como mostra a Figura 45.

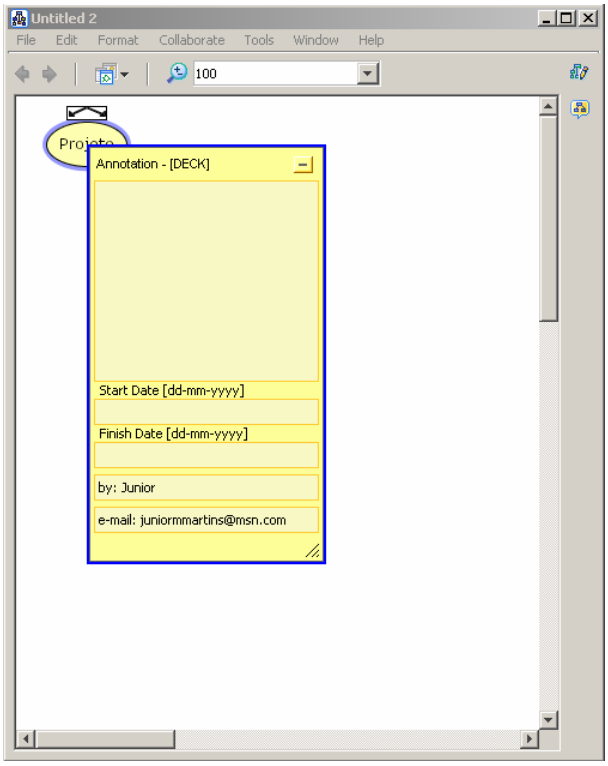


Figura 45 - Interface para captar informações de conceitos do tipo Deck

c) Transformação de um Mapa Conceitual formalizado em um cronograma

Foi criada uma interface para extrair um subconjunto de dados relativos aos Mapas Conceituais e a partir deles gerar um cronograma formal. Assim, nessa interface são exibidos todos os relacionamentos entre os conceitos do mapa atual, que são transformados no cronograma. Porém, os nodos do tipo "Adorno" não fazem parte da estrutura do cronograma, mas podem ser utilizados como anotações para descrever com mais detalhes algum conceito, sendo que esse processo faz apenas a leitura do Mapa Conceitual, uma vez que o mapa permanece o mesmo. Essa interface é apresentada na Figura 46.

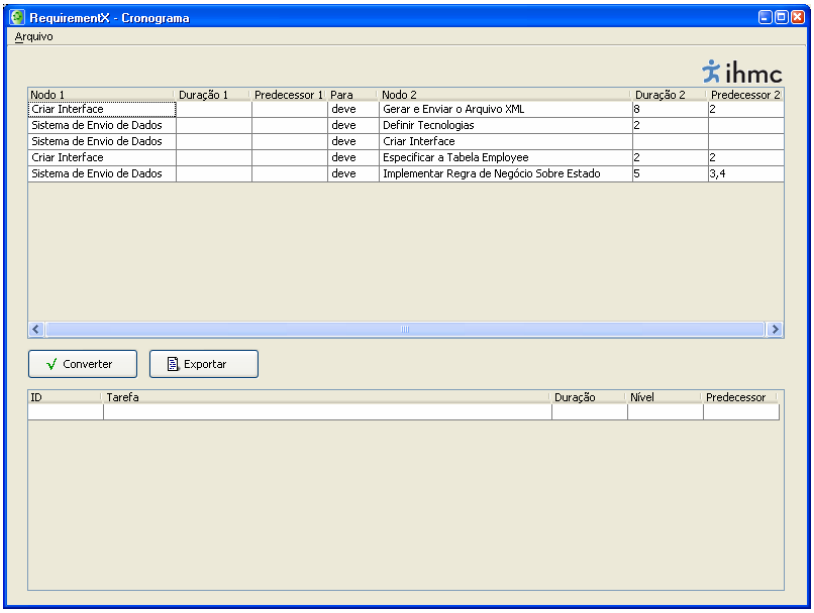


Figura 46 - Interface que converte uma EDT em um cronograma

4.4 Interoperabilidade dos requisitos com outras aplicações

Para prover uma interoperabilidade dos requisitos de *software* criados pelos usuários em formato de Mapa Conceitual (representando uma *User Story*), foi desenvolvida uma interface de exportação, conforme mostra a Figura 47.

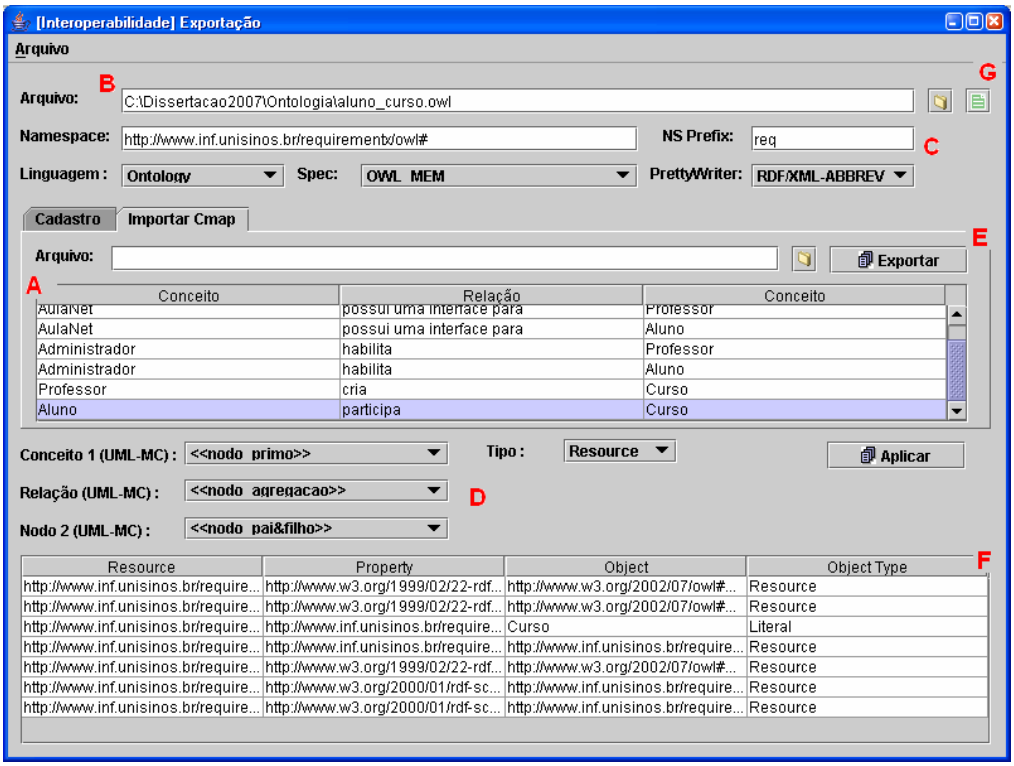


Figura 47 - Interface de exportação de Mapas Conceituais

A) Ao abrir a interface de exportação, o conteúdo do Mapa Conceitual (triplas) é carregado para a tabela da aba “Importar Cmap”. Assim, o Mapa Conceitual carregado na Figura 47 foi importado do Mapa Conceitual da Figura 59;

B) O usuário deverá definir o nome do arquivo OWL que será gerado;

C) Alguns parâmetros devem ser informados:

- *Namespace* – é o nome utilizado para identificar uma URI, sendo que, no XML, essa representação é feita por meio da propriedade *xmlns* [GON 2005];

- *NS Prefix* – nome da abreviação da *namespace*;

- Linguagem – o usuário poderá escolher se o arquivo exportado será um arquivo RDF ou um arquivo de ontologia;

- *Spec* - caso a linguagem escolhida for "*Ontology*", o usuário deverá definir qual a linguagem do perfil, por exemplo, "OWL_MEN" descreve um modelo utilizando o perfil OWL Full;

- *PrettyWrite* – é um método de escrita, que utiliza vantagens do RDF/XML para abreviar a sintaxe para gerar um *Model* mais compacto. Assim, para escrever grandes arquivos e preservar novos vazios, utilizamos o método *write* com o formato *N-Triples* [RDF 2006];

D) Modelar o Mapa Conceitual com a utilização da notação UML-MC [ROB 2003], ou seja, o usuário deve selecionar cada proposição do Mapa Conceitual, especificar os tipos e relações existentes e clicar no botão aplicar. Portanto, essa modelagem é necessária para incluir uma semântica ao arquivo exportado;

E) Clicar no botão exportar;

F) A ontologia exportada será exibida na tabela que fica na parte inferior da interface;

G) Ao clicar no botão “Visualizar Ontologia”, o arquivo gerado poderá ser visualizado, conforme mostra a Figura 48.

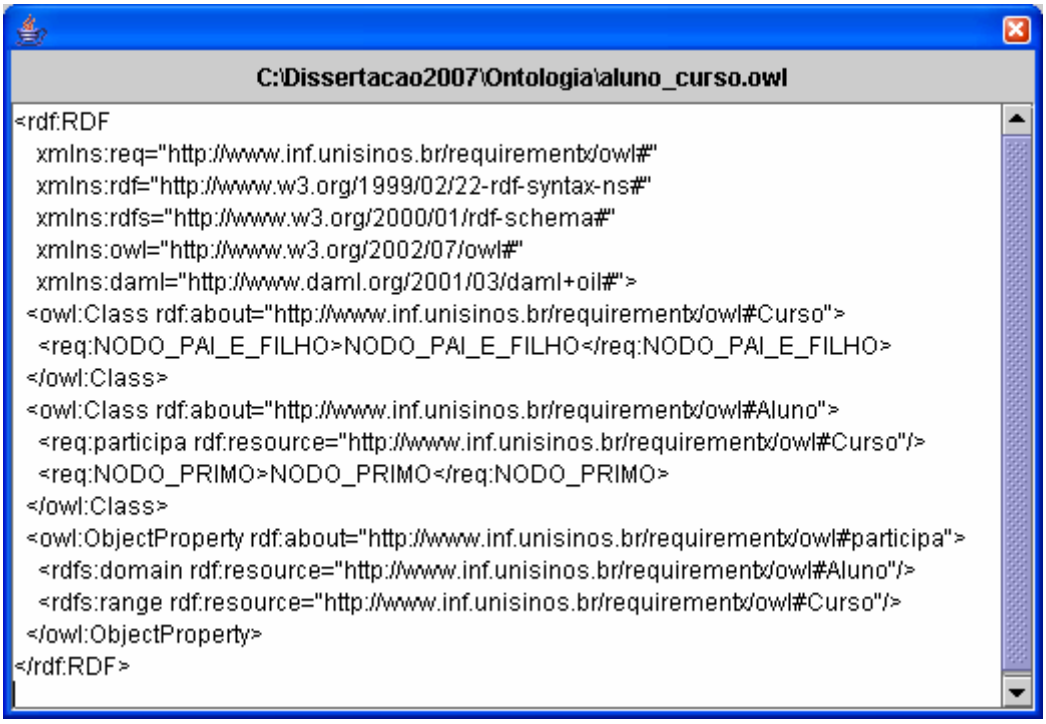


Figura 48 - Arquivo OWL gerado a partir de um Mapa Conceitual modelado com a notação UML-MC

A tradução do Mapa Conceitual modelado com a notação UML-MC para a linguagem OWL é realizada com a API Jena.

4.5 Processo de Elicitação

A ferramenta RequirementX proporciona uma formalização do processo e permite modelar e atualizar o conhecimento dos atores envolvidos rapidamente de forma cooperativa. Além disso, os Mapas Conceituais de requisitos de *software* serão utilizados para prover:

- Desmembramento da estrutura do trabalho (tarefas) para os desenvolvedores;
- Interoperabilidade com outros sistemas.

Além disso, o processo de elicitação e documentação de requisitos serão divididos em três etapas, conforme apresentado na Figura 49.

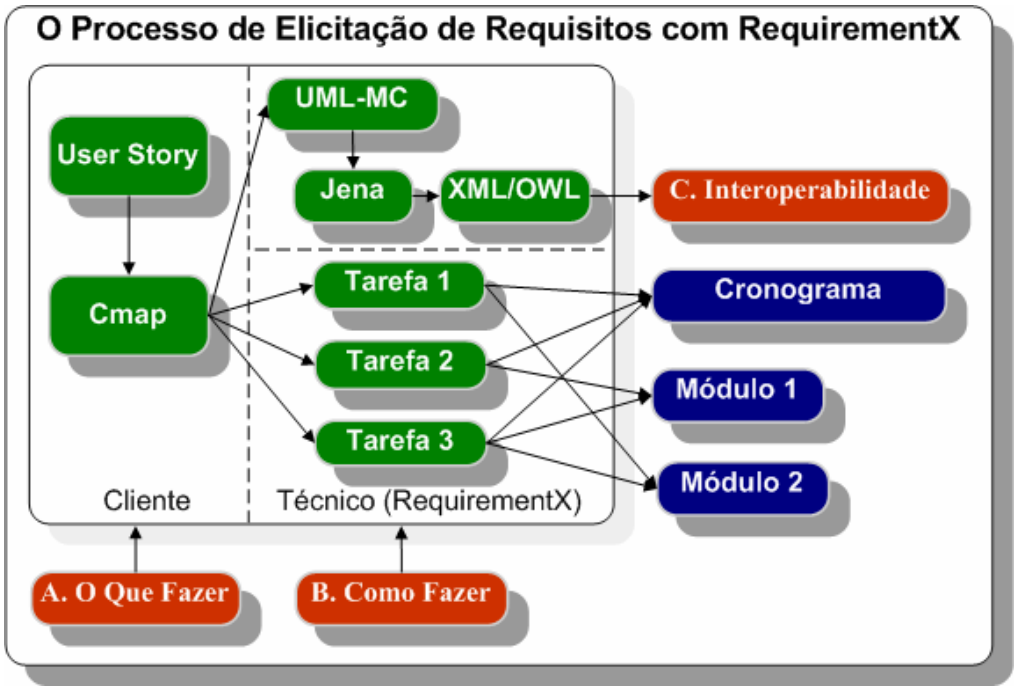


Figura 49 - Etapas do processo de elicitação de requisitos com o uso do RequirementX

Assim, para descrever cada uma das etapas da nossa proposta vamos realizar um simples processo de elicitação, cujo cenário é o seguinte: um setor de laminação de uma empresa siderúrgica produz barras de ferro que são utilizadas na construção civil. No entanto, esse setor está enfrentando alguns problemas na gerência da matéria prima utilizada na produção. É preciso então fazer um levantamento dos requisitos necessários para a criação de um sistema de informação para o gerenciamento da matéria prima.

A. O que fazer - em um primeiro momento, os usuários poderão construir Mapas Conceituais de uma forma cooperativa utilizando o *software* RequirementX, expressando o que deve ser construído. Dessa maneira, os usuários vão criar um Mapa Conceitual para cada objetivo, ou seja, cada Mapa Conceitual representará uma *User Story*, sendo que na Figura 50 podemos ver um exemplo de um Mapa Conceitual criado pelo usuário.

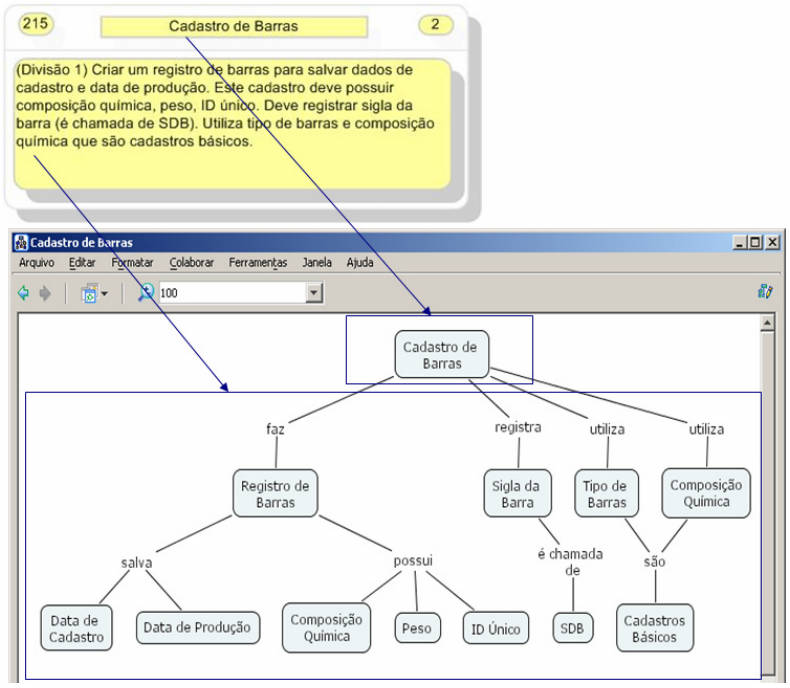


Figura 50 - Uma User Story representada por um Mapa Conceitual

Além disso, o usuário poderá utilizar os recursos da ferramenta para organizar os Mapas Conceituais e EDTs criados em diretórios separados, conforme apresentado na Figura 51. Os Mapas Conceituais poderão ser armazenados em diretórios locais no computador do usuário ou em servidores *CmapServer*.

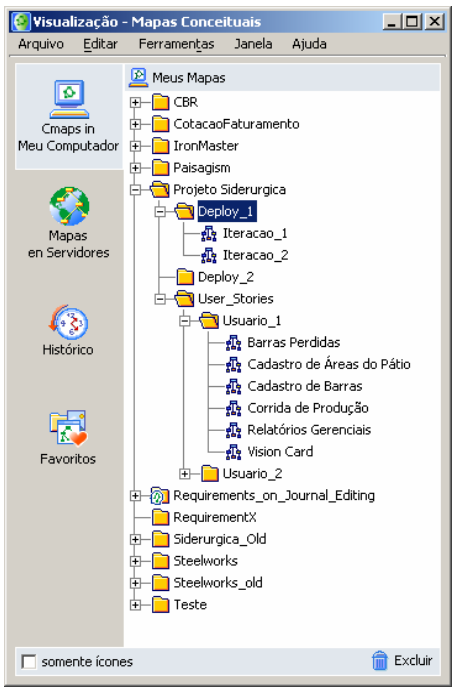


Figura 51 - Organizando os Mapas Conceituais em diretórios

B. Como fazer – Posteriormente, os desenvolvedores poderão criar uma estrutura de desmembramento de trabalho (EDT) com base nos Mapas conceituais criados pelos usuários e com o auxílio de algumas funcionalidades fornecidas pela ferramenta RequirementX.

Além disso, o desenvolvedor possui funcionalidades da ferramenta RequirementX que auxiliam na definição do tipo de conceito, cor, formato, armazenamento de informações específicas a tarefas e geração de um cronograma; as quais podem ser vistas com maiores detalhes na seção 4.3.

É importante também salientarmos os seguintes passos, que devem ser seguidos pelos desenvolvedores para a criação de uma EDT com a ferramenta RequirementX.

Passo 1 - Criar um novo Cmap (conforme mostra a Figura 52), sendo que na janela "*Window*" o usuário poderá criar a EDT, (maiores detalhes podem ser vistos na seção 4.2). É importante ainda deixar claro que utilizamos a mesma interface de criação de um Mapa Conceitual (Cmap) para a criação de uma EDT, embora uma EDT não seja um Mapa Conceitual, ou seja, a estrutura da Figura 54 é uma EDT (ou *WBS* em inglês) que aproveita o mesmo ambiente dos Mapas Conceituais para auxiliar o trabalho dos desenvolvedores.

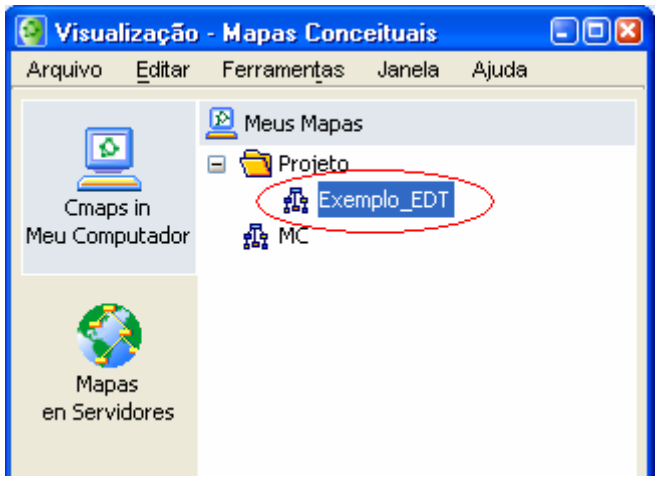


Figura 52 - Criando um novo Cmap para criar uma EDT

Passo 2 - Criar a EDT, definindo o tipo de cada conceito, sendo que para isso foram definidos tipos de conceitos para desmembrar um projeto em seus componentes e partes, auxiliando assim na identificação das tarefas. Os tipos de conceitos disponíveis podem ser vistos na Figura 53.

Deck	***Deck*** – representa o projeto, é representado por um formato oval e pela cor amarela, e pode ser definido automaticamente pela opção "Botão direito – Tipo de Nodo – Deck";
Pilha	**Pilha** – representa uma tarefa de resumo, é representada por um formato retangular e pela cor cinza, e pode ser definida automaticamente pela opção "Botão direito – Tipo de Nodo – Pilha";
User Story	***User Story*** – representa uma história do usuário que será transformada posteriormente em uma tarefa, é representada por um formato retangular de cantos arredondados e pela cor vermelha, e pode ser definida automaticamente pela opção "Botão direito – Tipo de Nodo – User Story";
Adorno	**Adorno** - são *User Stories* que acrescentam detalhes às outras *User Stories*, pilhas ou *decks*, é representado por um formato retangular de cantos arredondados e pela cor azul, e pode ser definido automaticamente pela opção "Botão direito – Tipo de Nodo – Adorno".

Figura 53 - Tipos de conceitos para representar uma EDT

Por sua vez, a Figura 54 apresenta um exemplo de uma EDT criada pelo desenvolvedor.

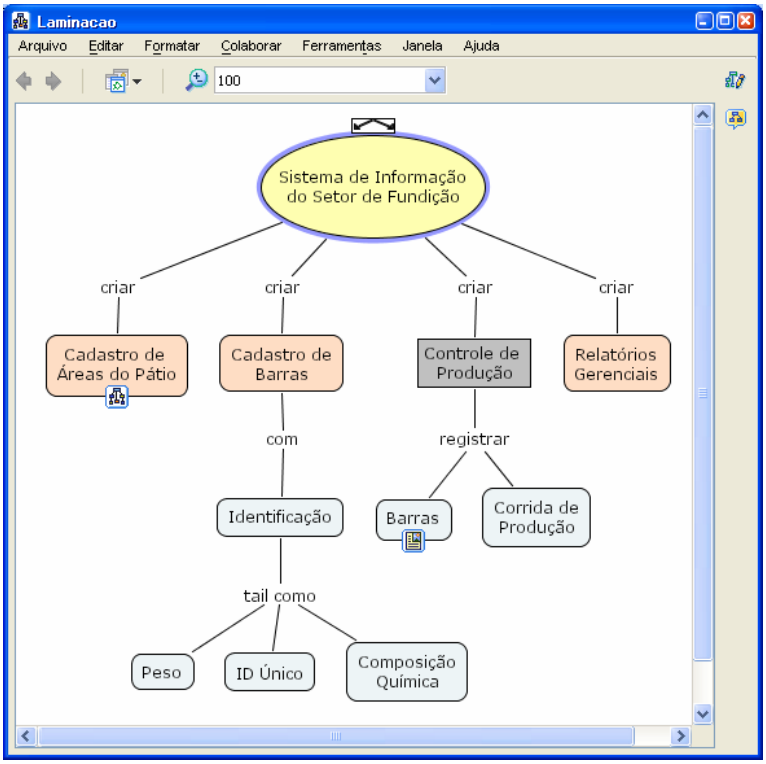


Figura 54 - Desmembramento das tarefas criado pelo desenvolvedor

Passo 3 - O próximo passo é adicionar informações necessárias para a construção do cronograma. Assim o nodo do tipo *Deck* deve receber as informações "Data Inicial" e "Data Final" do projeto, sendo que a interface da Figura 55 é utilizada para captar estas informações, que podem ser acessadas com a opção "Botão direito - Annotation RequirementX – Deck".

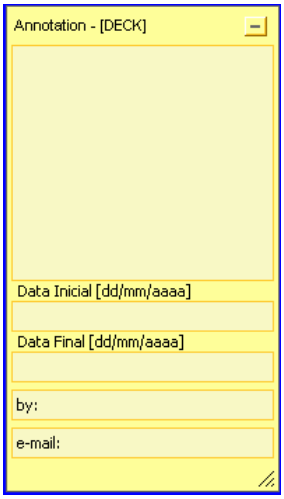


Figura 55 - Captação de informações referentes ao conceito do tipo Deck

Passo 4 - Os nodos do tipo *User Story* devem receber algumas informações que permitam a criação de um cronograma gerencial, conforme apresentado na Figura 56, sendo que podemos acessá-la com a opção "Botão direito - Annotation RequirementX – User Story". Além disso, é importante salientar que essas informações são muito importantes, pois servirão de base para a criação de estimativas e o controle de tarefas.

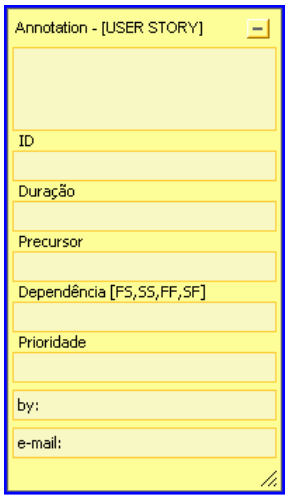
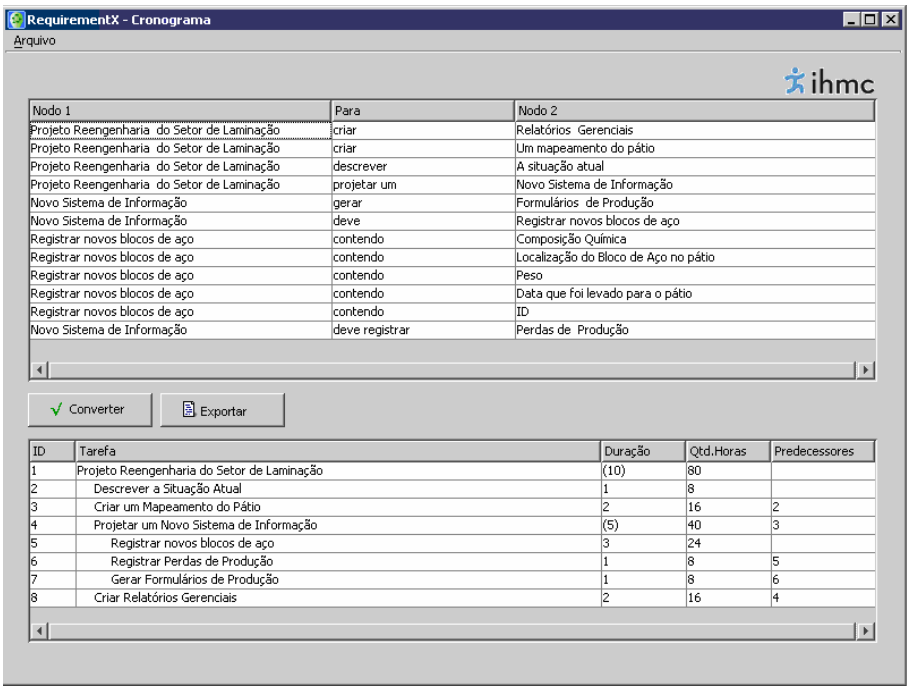


Figura 56 - Captação de informações referentes ao conceito do tipo User Story

Passo 5 - Após o desmembramento das tarefas pelo desenvolvedor, conforme exibido na Figura 54, pode-se transformá-lo em um cronograma. Assim, para essa função devemos utilizar a interface do *software* RequirementX, exibida na Figura 57, na qual o usuário pode visualizar em formato de tabela todo o desmembramento das tarefas. Além disso, ao clicar no botão "Converter", o *software* RequirementX transforma as informações do Mapa Conceitual em um cronograma formal.

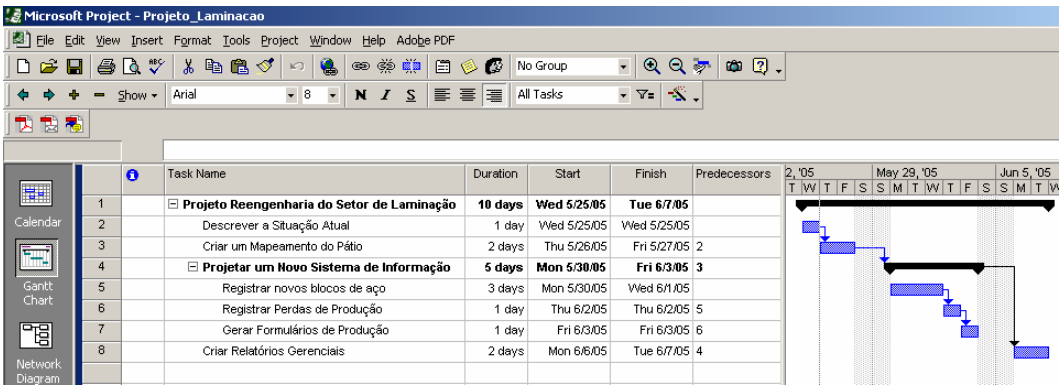


Nodo 1	Para	Nodo 2
Projeto Reengenharia do Setor de Laminação	criar	Relatórios Gerenciais
Projeto Reengenharia do Setor de Laminação	criar	Um mapeamento do pátio
Projeto Reengenharia do Setor de Laminação	descrever	A situação atual
Projeto Reengenharia do Setor de Laminação	projetar um	Novo Sistema de Informação
Novo Sistema de Informação	gerar	Formulários de Produção
Novo Sistema de Informação	deve	Registrar novos blocos de aço
Registrar novos blocos de aço	contendo	Composição Química
Registrar novos blocos de aço	contendo	Localização do Bloco de Aço no pátio
Registrar novos blocos de aço	contendo	Peso
Registrar novos blocos de aço	contendo	Data que foi levado para o pátio
Registrar novos blocos de aço	contendo	ID
Novo Sistema de Informação	deve registrar	Perdas de Produção

ID	Tarefa	Duração	Qtd.Horas	Predecessores
1	Projeto Reengenharia do Setor de Laminação	(10)	80	
2	Descrever a Situação Atual	1	8	
3	Criar um Mapeamento do Pátio	2	16	2
4	Projetar um Novo Sistema de Informação	(5)	40	3
5	Registrar novos blocos de aço	3	24	
6	Registrar Perdas de Produção	1	8	5
7	Gerar Formulários de Produção	1	8	6
8	Criar Relatórios Gerenciais	2	16	4

Figura 57 - Interface do software RequirementX para a geração de um cronograma

Por sua vez, o cronograma poderá ser exportado para um formato XML, para que possa ser utilizado por outras ferramentas. Assim sendo, na Figura 58 é apresentado um exemplo, onde as informações geradas pelo *software* RequirementX são exibidas no *software MS Project 2003*, em formato de tabela e gráfico Gantt.



	Task Name	Duration	Start	Finish	Predecessors
1	**Projeto Reengenharia do Setor de Laminação**	**10 days**	**Wed 5/25/05**	**Tue 6/7/05**	
2	Descrever a Situação Atual	1 day	Wed 5/25/05	Wed 5/25/05	
3	Criar um Mapeamento do Pátio	2 days	Thu 5/26/05	Fri 5/27/05	2
4	**Projetar um Novo Sistema de Informação**	**5 days**	**Mon 5/30/05**	**Fri 6/3/05**	**3**
5	Registrar novos blocos de aço	3 days	Mon 5/30/05	Wed 6/1/05	
6	Registrar Perdas de Produção	1 day	Thu 6/2/05	Thu 6/2/05	5
7	Gerar Formulários de Produção	1 day	Fri 6/3/05	Fri 6/3/05	6
8	Criar Relatórios Gerenciais	2 days	Mon 6/6/05	Tue 6/7/05	4

Figura 58 - Cronograma criado pelo software RequirementX sendo visualizado no MS Project

No final, teremos ainda a elicitação de requisitos realizada pelo usuário de um lado, e a formalização atribuída pelo engenheiro de *software* de outro. Ou seja, é separado "o que" o usuário quer fazer de "como" deve ser feito, porém estas informações estão disponíveis no ambiente da ferramenta RequirementX, sendo possível ainda visualizá-las e alterá-las com a ferramenta CmapTools que utiliza o mesmo formato de dados.

No entanto, é preciso muita atenção na construção dos requisitos com a utilização de Mapas Conceituais, principalmente na estruturação de pilhas e *User Stories*, pois uma estrutura de desmembramento de trabalho imprecisa e mal desenvolvida pode trazer vários problemas. Outro erro que pode trazer problemas é a omissão de *User Stories*

(tarefas) necessárias. Além do mais, é importante também definir bem o foco sobre a produção de um produto, dando à tarefa e à equipe um fim claro que facilita tanto a estimativa quanto o controle da tarefa. Finalmente, não podemos esquecer a regra básica de desmembrar o projeto em unidades de trabalho menores, que tenham um significado e que sejam gerenciáveis.

C. Interoperabilidade – Foi criado um mecanismo de exportação de requisitos (em formato de Mapas Conceituais) para arquivos OWL com a utilização da API Jena 2, sendo que essa exportação deve seguir os seguintes passos:

1) Modelar o Mapa Conceitual com a utilização da notação UML-MC [ROB 2003]. Assim, para demonstrar um exemplo, vamos utilizar o Mapa Conceitual da Figura 59.

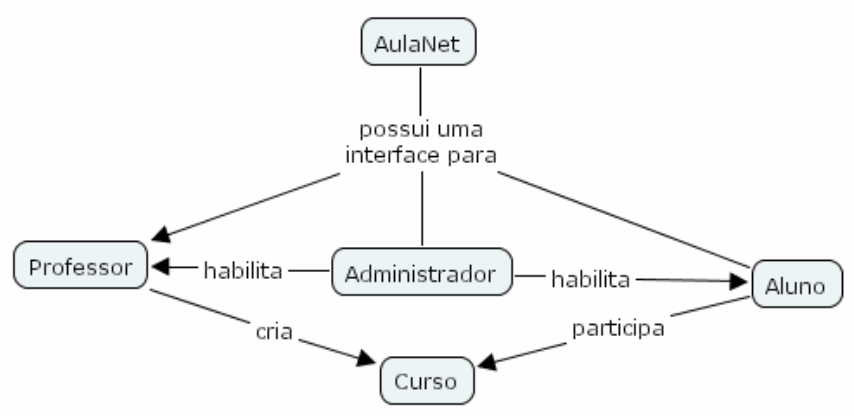


Figura 59 - Mapa conceitual AulaNet [ROB 2003]

Assim sendo, podemos interpretar esse Mapa Conceitual da seguinte maneira: o <<nodo_primo>> "Administrador" relaciona-se com os <<nodos_primo>> "Professor" e "Aluno", sendo que esses três conceitos encontram-se em um mesmo nível e são uma especificação do conceito AulaNet. Por sua vez, os <<nodos_primo>> "Professor" e "Aluno" relacionam-se com o <<nodo_pai&filho>> "Curso". Já o conceito "Administrador" não possui um conceito que seja subordinado a ele em um nível inferior.

Portanto, depois de identificado como representar os conceitos de um Mapa Conceitual, foi analisado como tratar as relações existentes entre os conceitos. Basicamente, os conceitos específicos representam uma parte do conceito geral, gerando uma relação do tipo todo/parte, sendo que a notação UML-MC possui dois estereótipos para representar relações entre conceitos, <<nodo_composicao>> e <<nodo_agregacao>>, que foram apresentados anteriormente nas seções 2.9.5 e 2.9.6 respectivamente.

Após interpretar o Mapa Conceitual e tratar suas relações, teremos o Mapa Conceitual modelado pela notação UML-MC, como mostra o exemplo da Figura 60.

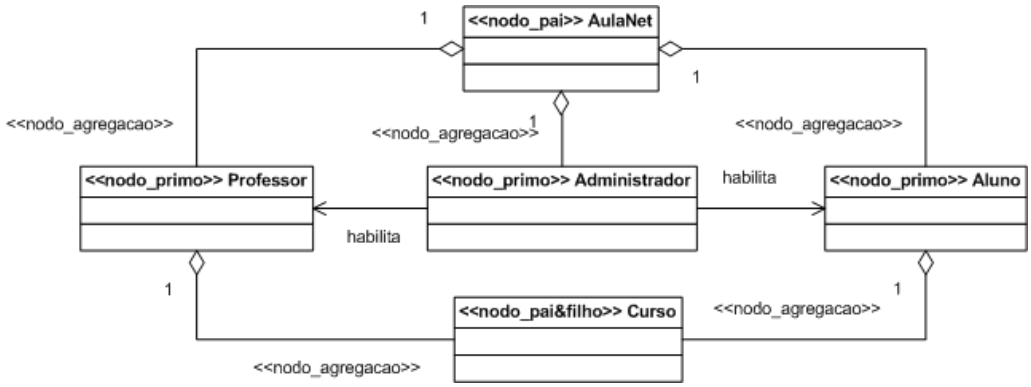


Figura 60 - Utilização dos estereótipos <<nodo_primo>> e <<nodo_agregacao>> [ROB 2003]

2) Traduzir o diagrama modelado com a notação UML-MC para OWL

Uma interface foi criada na aplicação RequirementX para realizar a conversão de UML-MC para OWL. Entretanto, essa conversão não é automática, pois antes dela o usuário deverá especificar os tipos e relações existentes no Mapa Conceitual. Ou seja, o usuário deverá interpretar o Mapa Conceitual e definir suas relações, sendo que a interface de conversão possui os mecanismos necessários para essa tarefa. Posteriormente, a interface fará o mapeamento da notação UML-MC para OWL com a utilização da API Jena.

Vamos, então, explicar como é o mapeamento da UML-MC para OWL [ONT 2006], sendo que para isso utilizaremos uma parte do diagrama apresentado na Figura 59, como mostra a Figura 61. Além disso, no capítulo 5 será realizado um exemplo prático baseado em um estudo de caso.

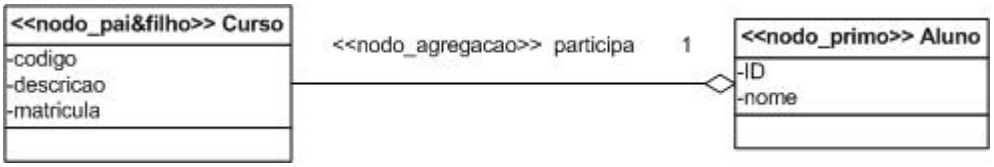


Figura 61 - Simples diagrama de classes modelado com UML-MC

Existem várias maneiras de realizar o mapeamento de UML para OWL, neste capítulo vamos apresentar um exemplo, embora ele pudesse ser mapeado de outras maneiras. Assim, vamos então descrever os detalhes do diagrama de classes da Figura 61, onde uma classe OWL é declarada pela atribuição de um nome a um tipo, como mostra o trecho de código abaixo:

```
<owl:Class rdf:ID=”Curso”>
```

Por sua vez, o relacionamento entre classes é chamado de propriedades (*properties*), e o relacionamento entre as classes “Curso” e “Aluno” é chamado de “participa”. Já a OWL é representada com o trecho de código abaixo:

```
<owl:ObjectProperty rdf:ID=”participa”>
```

Dessa forma, a relação entre classes em OWL é representada por *owl:ObjectProperty* e *owl:DatatypeProperty*, onde *domain* da propriedade é uma classe, e *range* é o tipo de classe. Já se o tipo da propriedade (*range*) é uma classe UML, devemos utilizar *owl:ObjectProperty*, caso contrário (por exemplo, um tipo de dado *string*) devemos utilizar *owl:DatatypeProperty*. A ferramenta RequirementX utiliza a relação de classes *owl:ObjectProperty* como padrão, mas permite que o usuário altere para *owl:DatatypeProperty* quando necessário. Podemos ver a tradução do modelo da Figura 61 na Tabela 8.

Classe	**Propriedade**	**Tipo da propriedade**	**Equivalente em OWL**
Curso	código	*CourseID*	<owl:ObjectProperty rdf:ID=”CodigoCurso”> <rdfs:domain rdf:resource=”Curso”/> <rdfs:range rdf:resource=”CourseID”/> </owl:ObjectProperty>
	descrição	*string*	<owl:DatatypeProperty rdf:ID=”DescricaoCurso”> <rdfs:domain rdf:resource=”Curso”/> <rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#string”/> </owl:ObjectProperty>
	matrícula	*integer*	<owl:DatatypeProperty rdf:ID=”MatriculaCurso”> <rdfs:domain rdf:resource=”Curso”/> <rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#integer”/> </owl:ObjectProperty>
Aluno	ID	*StudentIdent*	<owl:ObjectProperty rdf:ID=”IDAluno”> <rdfs:domain rdf:resource=”Aluno”/> <rdfs:range rdf:resource=” StudentIdent”/> </owl:ObjectProperty>
	nome	*string*	<owl:DatatypeProperty rdf:ID=”NomeAluno”> <rdfs:domain rdf:resource=”Aluno”/> <rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#string”/> </owl:ObjectProperty>

Tabela 8 - Tradução de UML para OWL [ONT 2006]

Ainda, um exemplo de tradução de um relacionamento UML para OWL pode ser visto na Tabela 9.

Relacionamento	Classe 1	Classe 2	Equivalente em OWL
Participa	Curso	Aluno	<owl:ObjectProperty rdf:ID=”participa”> <rdfs:domain rdf:resource=”Curso”/> <rdfs:range rdf:resource=”Aluno”/> </owl:ObjectProperty>

Tabela 9 - Tradução de um relacionamento UML para OWL [ONT 2006]

4.6 Comparando a RequirementX com as outras abordagens

Tradicionalmente, as *User Stories* são escritas manualmente em cartões indexados que são facilmente armazenados, exibidos, organizados e distribuídos entre a equipe. Entretanto, a maioria dos outros produtos de trabalho de uma equipe de desenvolvimento de *software* apresenta um formato eletrônico, e o desenvolvimento de *software* ágil está cada vez mais sendo adotado por equipes que trabalham em locais remotos. Portanto, nessas circunstâncias, as equipes de desenvolvimento procuram por soluções de *software* para criar e gerenciar *User Stories* de uma forma eficiente e compartilhada.

Sendo assim, prover suporte à XP através de uma ferramenta, para complementar ou substituir métodos existentes, pode impactar o ambiente de desenvolvimento. Por exemplo, a utilização de uma nova ferramenta pode impor restrições ao fluxo normal de trabalho ou mudar de forma negativa processos já existentes de elicitação de requisitos. A respeito disso, o maior objetivo das propostas apresentadas é ajustá-las ao contexto de projetos XP sem violá-lo.

Por essa razão, as abordagens apresentadas têm focado principalmente no gerenciamento de *User Story* e no CSCW. Além disso, essas soluções, com exceção feita à Web-Desktop, focam no suporte de tarefas específicas, especialmente no que diz respeito ao aumento da colaboração da equipe em um ambiente XP.

Entretanto, as maiores desvantagens encontradas nessas abordagens foram as seguintes:

- O uso de *NetMeeting* que restringe o usuário a utilizar a plataforma Microsoft Windows;
- Redução de capacidade da rede em virtude das conferências;
- Aplicações *web* suscetíveis a problemas de resolução e conflitos de atualização, tais como Wiki;
- Falta de um mecanismo adequado para colaboração síncrona na elicitação de *User Stories* durante o Jogo de Planejamento;
- Questões de internacionalização tais como *layouts* de teclados e caracteres de diferentes países.

Por outro lado, essas abordagens apresentaram bons exemplos de gerenciamento de *User Stories* e trabalho cooperativo suportado por computador, assim como outras atividades de XP, que ajudaram muito na melhoria da nossa proposta.

Sendo assim, foi realizada uma comparação da ferramenta RequirementX com as outras ferramentas através do cruzamento de algumas características, conforme apresentado na Figura 62.

Características	RequirementX	PAM	XPSwiki	DotStories	MILOS	TUKAN	Web-Desktop	Storymanager
Suporte à *User Story*	S	S	S	S	S	S	S	S
Suporte a tarefas	S	S	S		S			S
Suporte a trabalho cooperativo	S	S	S	N	S	S	S	
Recursos visuais poderosos	S	S		S				N
Facilidade de uso	S							
Realizado estudo de caso	S	S						S
Tipo de ambiente	D	D	W	W	W	W	W	D
Possui ID automático	N							S
Possui sistema de versões	S		S					
Possui persistência	S	S	S					
Suporte a mútiplos projetos	S		S	S				
Suporte a teste unitário	N	S	N					
Suporte a teste de aceitação	N	N	S					
Provê *Chat*	N	S				S	S	
Suporte à *Planning Game*	S	S	S		S	S	S	
Possui organização hierárquica	S	S	S	S				
Suporte à internacionalização	S						N	
Utiliza mapas conceituais	S	N	N	N	N	N	N	N

Legenda:

S	Possui a característica.
N	Não possui a característica.
D	Aplicação Desktop.
W	Aplicação Web.
	Informação não encontrada.

Figura 62 - Comparando RequirementX com as outras abordagens

Assim, apesar de não encontrarmos informações para algumas características, essa comparação serviu para destacar as vantagens da ferramenta RequirementX em relação às outras ferramentas, dentre as quais destacamos:

- Utilização de Mapas Conceituais - propondo-se o uso de Mapas Conceituais na descoberta, gerência e representação dos requisitos de clientes no desenvolvimento de *software*, ocorre uma participação mais ativa e uma melhor comunicação das pessoas envolvidas, sendo a comunicação uma das palavras chaves das metodologias ágeis de desenvolvimento. Além disso, também aproveitamos a liberdade na seleção de conceitos e frases de ligação, pois acreditamos que esse é um dos pontos fortes da ferramenta [DER 2004] [COF 2003] [CAN 2002] [COF 2002]. Isso significa que, quando conceitos e relações são escolhidos com cuidado, os mapas tornam-se ferramentas poderosas para observar a nuance dos significados, e que seu rico potencial de expressão vem da capacidade de cada mapa em permitir ao seu criador o uso de um número virtualmente ilimitado de frases de ligação para definir o significado da relação entre conceitos;
- Mecanismos de exportação - quando o usuário conclui a confecção do Mapa Conceitual, ele tem a oportunidade de exportar o seu trabalho para diferentes formatos, como por exemplo, um documento *web*, uma figura ou um arquivo XML [BEU 2003] e [CAB 2000];
- Interoperabilidade - RequirementX possui um mecanismo para prover interoperabilidade com outros sistemas, através de um mecanismo de exportação de requisitos (em forma de Mapas Conceituais) para arquivos OWL;
- Recursos visuais - os Mapas Conceituais são instrumentos facilitadores no processo de elicitação porque possibilitam uma visão gráfica de um determinado domínio de problema e seus relacionamentos. De acordo com Ontoria [ONT 1993], é de fato um bom instrumento para detectar com rapidez a quantidade e qualidade das informações em determinado momento, uma vez que se evidencia com grande clareza o número de conceitos relevantes para os usuários, como também a estruturação dos mesmos. Assim, o RequirementX possibilita que os usuários enriqueçam os seus trabalhos, já o enriquecimento é caracterizado por mídias, como URLs, vídeos, sons, e mapas. Portanto, para sinalizar que o Mapa Conceitual apresenta uma mídia anexada em um determinado conceito, criam-se ícones junto com os conceitos;
- Ferramenta amigável - segundo Cañas [CAC 2002], a ferramenta em questão é amigável e de fácil compreensão, pois, no momento em que o indivíduo está elaborando o seu mapa, é disponibilizada a possibilidade de alterar o estilo do seu trabalho, através da modificação dos atributos, como a cor, o estilo, e o tamanho da fonte, e a cor do preenchimento da caixa e das bordas;
- Suporte à colaboração - RequirementX permite o compartilhamento dos mapas com outros usuários através da ferramenta *CmapServer*;
- Internacionalização – RequirementX provê suporte em vários idiomas.

O capítulo a seguir apresenta um estudo de caso desenvolvido para a elicitação de requisitos de *software* com *Extreme Programming* (XP) utilizando a ferramenta RequirementX.

5 Estudo de caso

Neste capítulo, vamos descrever um estudo de caso da ferramenta proposta. No entanto, antes de iniciar devemos verificar se os usuários e desenvolvedores envolvidos possuem o conhecimento necessário em Mapas Conceituais e *Extreme Programming*, pois, caso o conhecimento dos usuários e/ou desenvolvedores seja insuficiente, é necessário realizar um treinamento para suprir essa necessidade.

5.1 Cenário do estudo de caso

O estudo de caso será baseado em uma regulamentação do governo americano que uma empresa deve seguir. Para isso é necessário construir uma interface de *software* para o envio de dados de empregados, eximindo a empresa de uma despesa mensal com o envio manual dessas informações.

Primeiramente, algumas perguntas simples foram feitas ao cliente para um melhor entendimento do cenário:

- O que deve ser feito?
    - Enviar informações requeridas pelo governo americano da base de dados da empresa (através de uma interface de *software*) para o *American Job Bank* (Banco de Empregos Americano).
- Onde?
    - Da base de dados da empresa para o *American Job Bank*, sendo que os dados requeridos são alguns campos da base de dados, mais especificamente da tabela *EMPLOYEE*, que possui a estrutura da Tabela 10.

employee_id	NUMBER(12)
location1	VARCHAR2(100)
state	VARCHAR2(2)
job_level	NUMBER(5)
job_desc	VARCHAR2(150)
description	VARCHAR2(300)
company_id	VARCHAR2(30)

Tabela 10 - Estrutura da tabela *EMPLOYEE*

- Quando?
    - Até 15 de janeiro de 2007.
- Por quê?
    - Devido a uma regulamentação do governo americano, todos os empregados com uma determinada posição na empresa devem ter seus dados postados na base do *American Job Bank*. Atualmente, esse processo está sendo feito manualmente na empresa com um custo mensal

de cinco mil dólares. Então, para eliminar esse custo e estar de acordo com as regulamentações do governo americano, uma interface de *software* precisa ser criada para automatizar o processo de envio de dados.

- Como fazer?
    - Criar uma interface que envie um arquivo XML com todos os dados extraídos da base de dados da empresa para o AJB, sendo que quem vai fornecer todos os detalhes necessários para o desenvolvimento dessa interface são os usuários através de *User Stories* em formato de Mapas Conceituais.

5.2 Primeira elicitação de requisitos

A seguir, serão apresentados alguns Mapas Conceituais criados pelo usuário, sendo que cada Mapa Conceitual representa uma *User Story*.

O primeiro Mapa Conceitual criado foi sobre a definição das tecnologias utilizadas no desenvolvimento da interface de envio de dados dos empregados, conforme mostra a Figura 63.

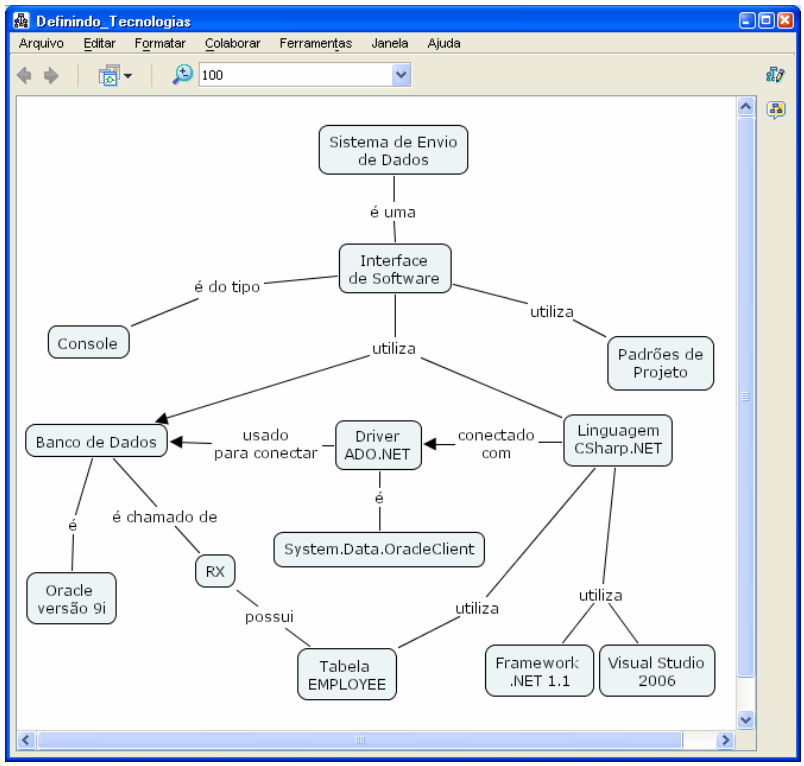


Figura 63 - Mapa conceitual sobre a definição das tecnologias

Assim, nesse mapa já podemos obter várias informações importantes para o desenvolvimento da interface, como, por exemplo, que será uma interface do tipo console, cuja linguagem será *CSharp.NET*, e seu banco de dados será *Oracle* versão 9i.

O segundo Mapa Conceitual (Figura 64) especifica os detalhes sobre a tabela *EMPLOYEE*, ou seja, quais campos devem ser selecionados, quais regras de negócio estão envolvidas e quais são os detalhes sobre conexão com o banco de dados.

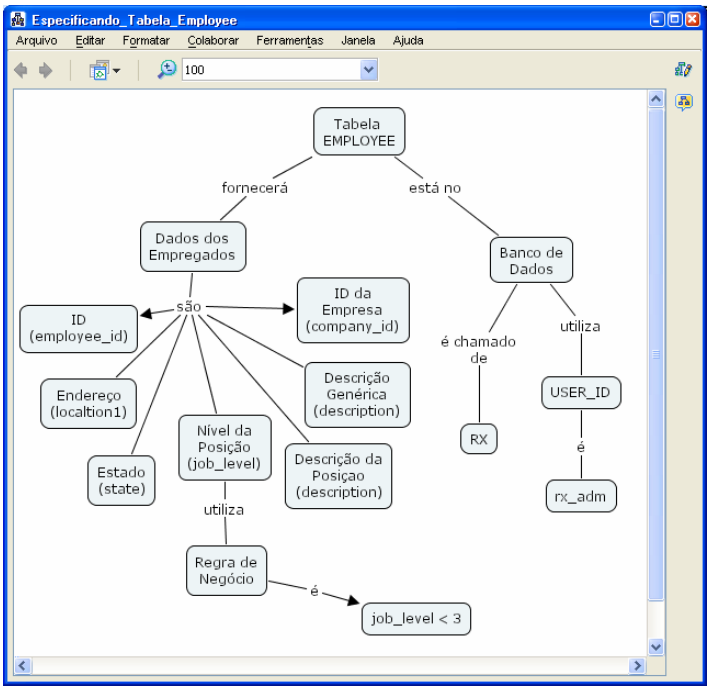


Figura 64 - Mapa conceitual com especificações sobre a tabela EMPLOYEE

A partir daí, podemos verificar que a empresa possui níveis de posições que vão de 1 a 7, sendo que quanto menor o nível, maior o cargo na empresa. É possível notar também que a regra de negócio definida pelo usuário define que somente registros de posições com níveis 1 e 2 devem ser enviados.

Já o terceiro Mapa Conceitual (Figura 65) apresenta detalhes sobre a geração e o envio dos dados em formato XML.

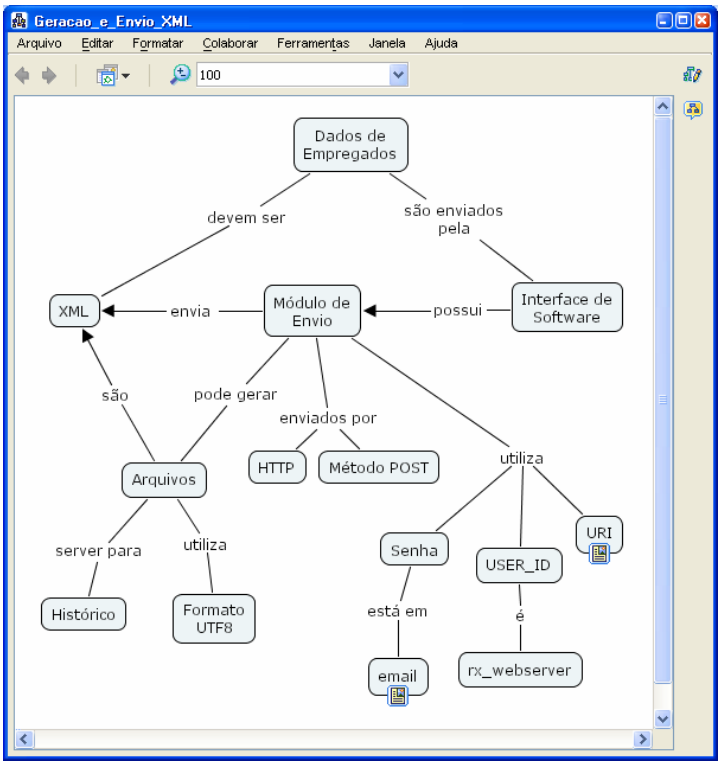


Figura 65 - Mapa conceitual sobre o arquivo XML

Podemos perceber assim, que um detalhe importante nesse requisito é que a interface de *software* não necessita persistir um arquivo XML porque a geração e envio das informações é realizada em memória. Porém, como o usuário definiu na proposição "Módulo de Envio -> pode gerar -> Arquivos", um parâmetro deverá ser criado na interface para permitir ou não a geração de um arquivo XML após o envio dos dados, sendo que esses arquivos podem ser utilizados como histórico.

Um quarto Mapa Conceitual (Figura 66) foi criado para descrever uma regra de negócio encontrada posteriormente pelo usuário, no qual o nome da cidade e o código postal devem ser incluídos no arquivo XML, mas, como a tabela fornece apenas o estado (*state*) do empregado, será necessário criar uma regra de negócio para tratar dessa necessidade.

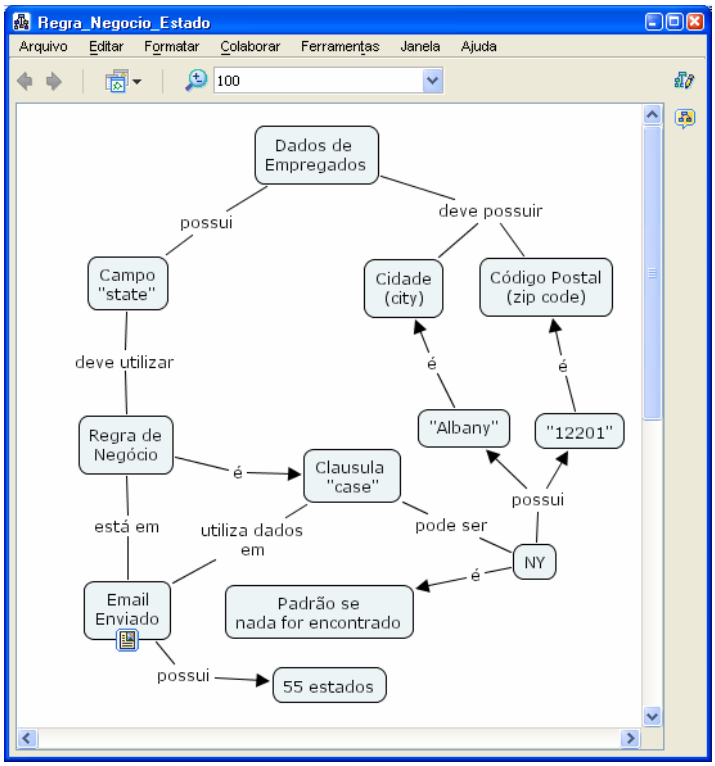


Figura 66 - Regra de negócio de estado

Podemos ver assim que o documento vinculado ao conceito “Email Enviado” possui uma tabela com a capital e o CEP de cada estado. Por exemplo, se o estado é FL (Flórida) a cidade será “Tallahassee” e o CEP será “32301”. No entanto, caso o registro atual não possua o estado (ou seja um estado incorreto) o sistema deve utilizar o estado NY (Nova York), cuja capital é "Albany", e cujo CEP é "12201", uma vez que NY é onde está localizada a matriz da empresa. Entretanto, é importante dizer que a relação dos estados com suas respectivas capitais e CEPs não foi descrita no Mapa Conceitual porque o país possui 55 estados.

5.2.1 Planejamento e decomposição

Após a criação dos requisitos, os desenvolvedores podem criar uma estrutura de desmembramento de trabalho (EDT) utilizando as funcionalidades da ferramenta RequirementX, como apresentado na Figura 67.

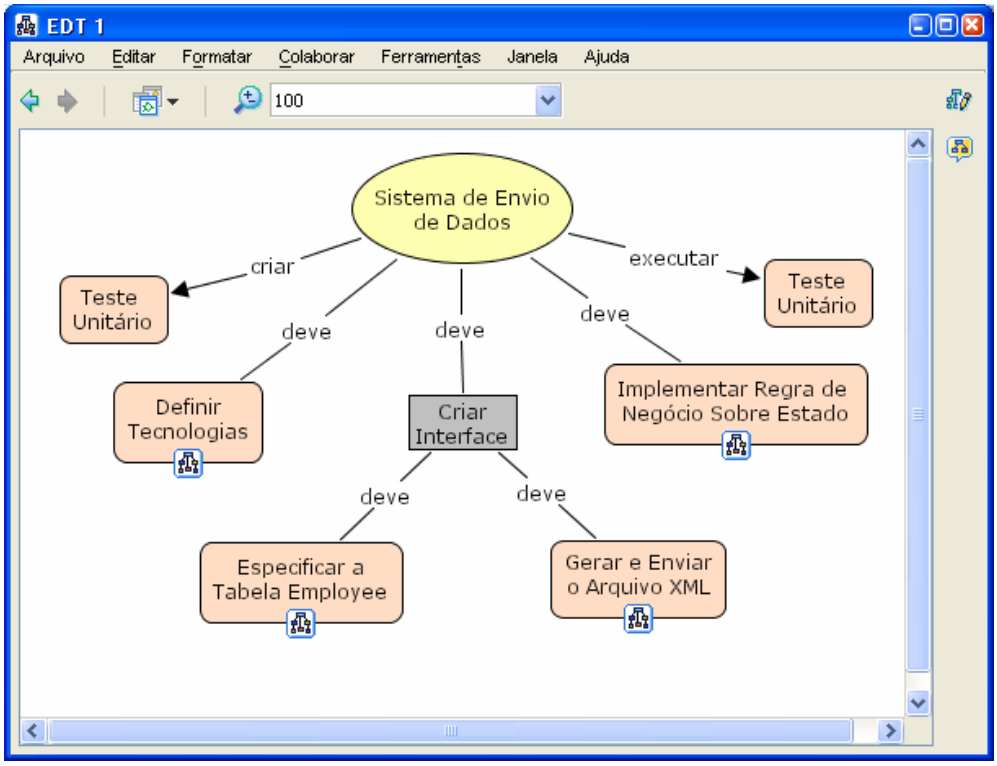


Figura 67 - Estrutura de desmembramento de trabalho criada no primeiro release

Entretanto, devemos lembrar que essa estrutura EDT criada não é um Mapa Conceitual. Além do mais, um detalhe importante que vale a pena destacar é que podemos vincular aos nodos do tipo *User Story* (tarefas) o Mapa Conceitual criado para essa tarefa.

Além disso, duas atividades foram incluídas para a criação e execução dos testes unitários, sendo que as informações sobre o projeto devem ser obtidas através da opção "Botão Direito sobre o nodo Deck – Annotation RequirementX – Deck", conforme apresentado na Figura 68.

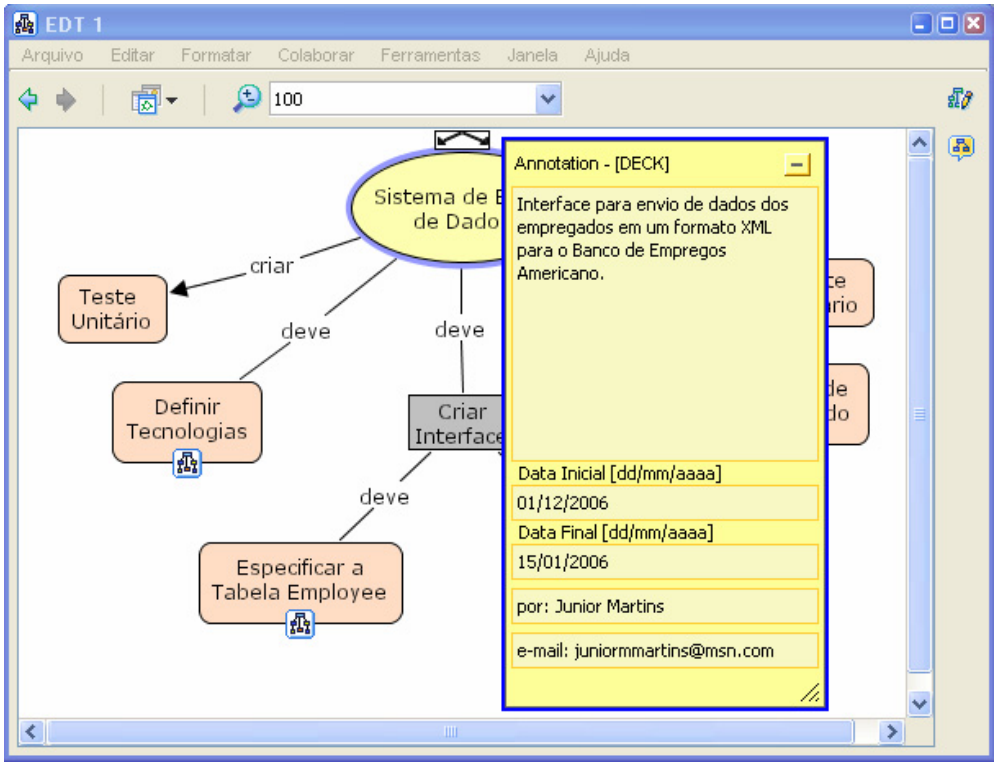


Figura 68 - Obtendo informações do nodo tipo Deck (projeto)

Por sua vez, as informações sobre cada nodo do tipo *User Story* (cada tarefa) devem ser obtidas através da opção "Botão Direito sobre o nodo User Story – Annotation RequirementX – User Story", conforme apresentado na Figura 69, sendo que a estimativa é definida pelos desenvolvedores e a prioridade pelos usuários.

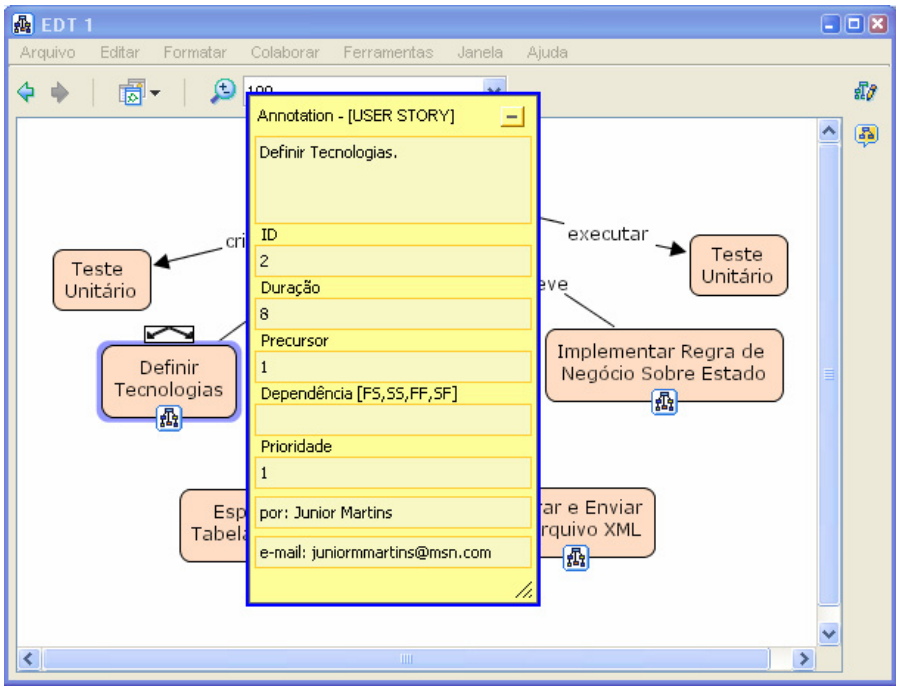
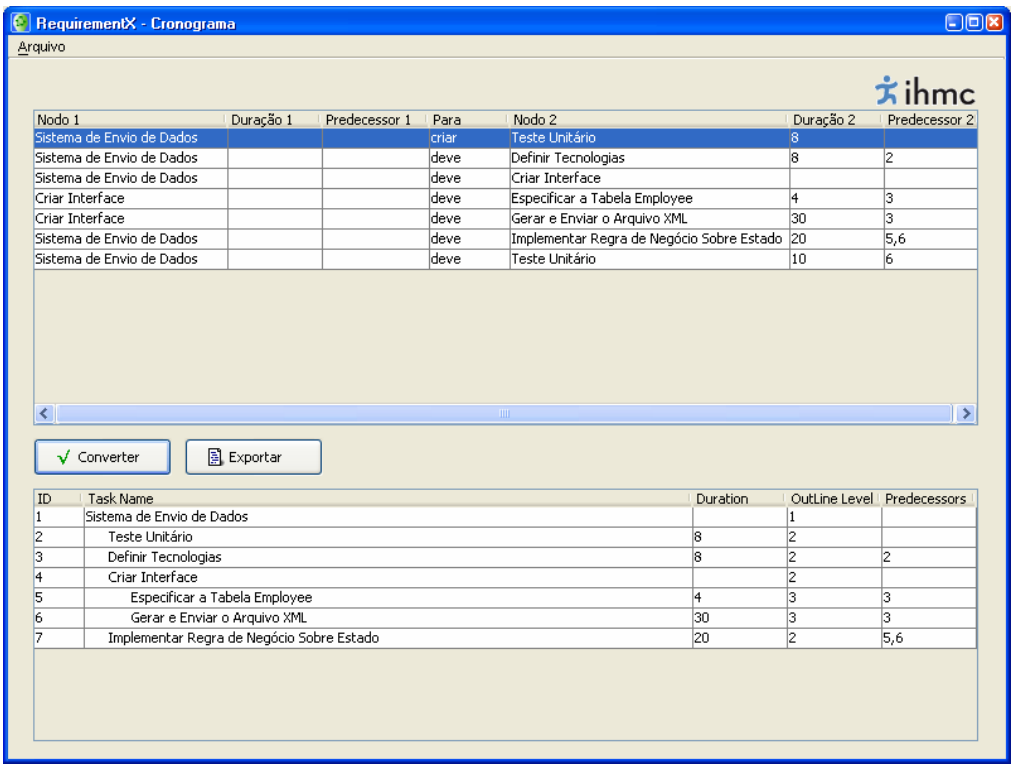


Figura 69 - Obtendo informações do nodo tipo User Story (tarefas)

Posteriormente, após os desenvolvedores estimarem todas as tarefas (utilizando a interface exibida na Figura 69) e atribuírem suas prioridades (definidas pelos usuários), é possível gerar automaticamente um cronograma através da janela "RequirementX – Cronograma" (Figura 70), disponível no item de menu "Ferramentas – Gerar Cronograma".



Nodo 1	Duração 1	Predecessor 1	Para	Nodo 2	Duração 2	Predecessor 2
Sistema de Envio de Dados			criar	Teste Unitário	8	
Sistema de Envio de Dados			deve	Definir Tecnologias	8	2
Sistema de Envio de Dados			deve	Criar Interface		
Criar Interface			deve	Especificar a Tabela Employee	4	3
Criar Interface			deve	Gerar e Enviar o Arquivo XML	30	3
Sistema de Envio de Dados			deve	Implementar Regra de Negócio Sobre Estado	20	5,6
Sistema de Envio de Dados			deve	Teste Unitário	10	6

ID	Task Name	Duration	OutLine Level	Predecessors
1	Sistema de Envio de Dados		1	
2	Teste Unitário	8	2	
3	Definir Tecnologias	8	2	2
4	Criar Interface		2	
5	Especificar a Tabela Employee	4	3	3
6	Gerar e Enviar o Arquivo XML	30	3	3
7	Implementar Regra de Negócio Sobre Estado	20	2	5,6

Figura 70 - Interface para geração automática de um cronograma

Assim, ao abrir a janela da Figura 70, teremos duas tabelas: a primeira será carregada com a EDT criada pelos desenvolvedores e as informações obtidas com a interface da Figura 67. Por sua vez, ao clicar no botão "Converter", a segunda tabela será carregada com um cronograma, sendo que para a obtenção e manipulação dos dados, utilizamos a programação Java (padrão da arquitetura RequirementX), e trabalhamos com as informações em formato de proposições, como mostra a Figura 71, para posteriormente carregarmos e manipularmos as informações em vetores. Isso significa que estamos utilizando as mesmas funcionalidades de manipulação de Mapas Conceituais para manipular estruturas EDT porque ambas utilizam a estrutura de "conceito-relação-conceito".

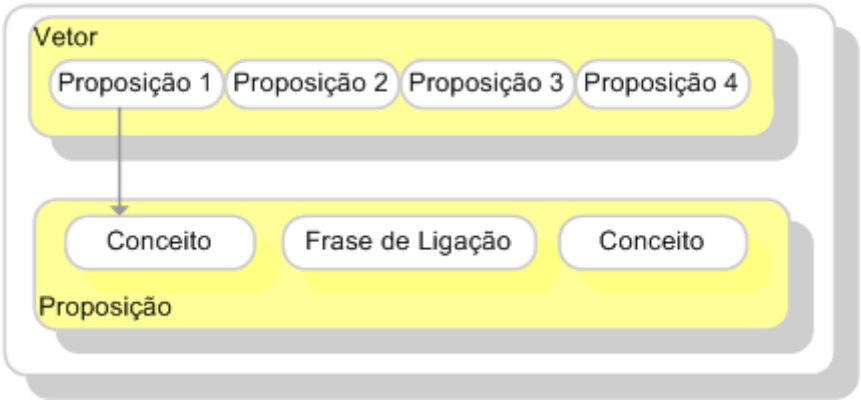


Figura 71 - Estrutura de uma proposição na arquitetura RequirementX

Podemos verificar também que ao clicar no botão "Exportar", um arquivo XML será criado e poderá ser importado pelo *software MS Project 2003*, como apresentado na Figura 72.

	Nome da tarefa	Duração	Início	Término	Predecessoras
1	**Sistema de Envio de Dados**	**8,25 dias**	**Sex 1/12/06**	**Qua 13/12/06**	
2	Teste Unitário	8 hrs	Sex 1/12/06	Sex 1/12/06	
3	Definir Tecnologias	8 hrs	Seg 4/12/06	Seg 4/12/06	2
4	**Criar Interface**	**3,75 dias**	**Ter 5/12/06**	**Sex 8/12/06**	
5	Especificar a Tabela Employee	4 hrs	Ter 5/12/06	Ter 5/12/06	3
6	Gerar e Enviar o Arquivo XML	30 hrs	Ter 5/12/06	Sex 8/12/06	3
7	Implementar Regra de Negócio Sobre Estado	20 hrs	Sex 8/12/06	Qua 13/12/06	5;6

Figura 72 - Cronograma gerado automaticamente pela ferramenta RequirementX

O próximo passo foi planejar as iterações e *releases*, sendo que foi definido que serão necessárias duas iterações até chegar a um *release* final. Assim, a primeira iteração trabalha com os seguintes requisitos:

- Definir tecnologias;
- Especificar a tabela *EMPLOYEE*;
- Gerar e enviar o arquivo XML.

Ou seja, com as três primeiras *User Stories* já é possível obter um resultado concreto, uma vez que já conseguiremos realizar o envio das informações dos empregados.

Já em uma segunda iteração, o seguinte requisito será implementado, chegando assim a uma versão final da aplicação:

- Implementar regra de negócio sobre estado.

Além disso, os usuários e desenvolvedores podem utilizar os recursos da ferramenta RequirementX para organizar os Mapas Conceituais e EDTs em pastas, conforme mostra a Figura 73.

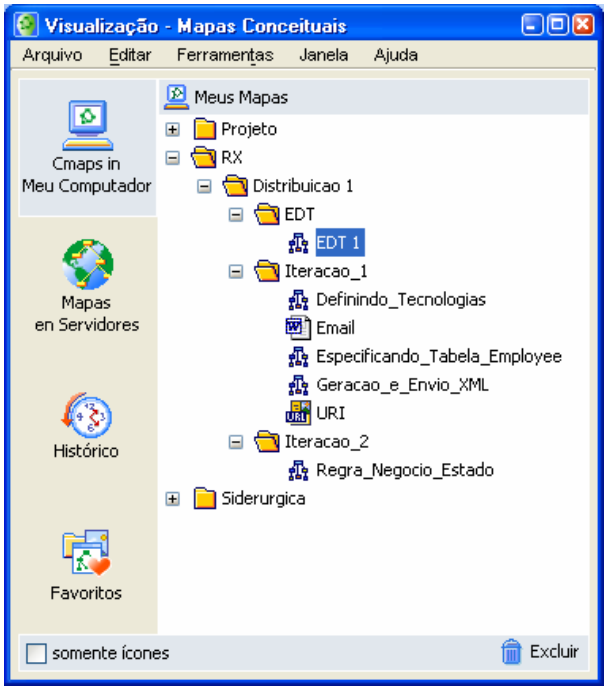


Figura 73 - Organização das User Stories criadas na distribuição 1

5.2.2 Implementação

Com a implementação da primeira iteração tivemos o seguinte resultado:

- Criação dos testes unitários;
- Uma tabela temporária chamada *RX_RESULT* foi criada com a mesma estrutura da tabela *EMPLOYEE*, a qual será utilizada para armazenar os dados antes de gerar o arquivo XML;
- A interface estará disponível através de um arquivo executável chamado de "RX_Interface.exe";
- A interface possui alguns parâmetros, de acordo com a sintaxe abaixo:

  RX_Interface.exe [/A dir_xml]; onde:

/A	Cria um arquivo de XML com os dados enviados via HTTP pela interface de *software*.
dir_xml	Determina o diretório do arquivo XML.
/?	Apresenta uma mensagem de ajuda explicando como funciona a interface, mas não executa a aplicação.

Tabela 11- Parâmetros da interface criada no estudo de caso

- O trecho de código da Figura 74 é um exemplo de como a interface criada no estudo de caso gera os dados XML em memória para posteriormente enviá-los.

```
1 XmlNode node;
2 XmlNode txtNode;
3
4 XmlDocument xmlDoc = new XmlDocument();
5 XmlDeclaration decl = xmlDoc.CreateXmlDeclaration("1.0", "utf-8", "");
6 XmlDoc.InsertBefore(decl, xmlDoc.DocumentElement);
7
8 XmlNode pNode = xmlDoc.CreateElement("Requisition");
9
10 node = xmlDoc.CreateElement("RequisitionNo");
11 txtNode = xmlDoc.CreateTextNode(requisition["state"].ToString());
12 node.AppendChild(txtNode);
13 pNode.AppendChild(node);
14 ...
15 XmlNode newNode = xmlDoc.CreateElement("Requisitions");
16 newNode.AppendChild(pNode);
17 xmlDoc.AppendChild(newNode);
18 return xmlDoc.OuterXml;
```

Figura 74 - Exemplo de como a interface do estudo de caso gera dados XML em memória

Assim, podemos descrever o código da Figura 74 da seguinte maneira:

Linhas 1 e 2: declaração de variáveis temporárias;

Linhas 4 e 5: declaração de um documento XML em memória;

Linha 8: criação do nodo principal;

Linha 10: criação de um nodo filho;

Linha 11: atribuição do valor do campo "*state*" da tabela RX para a variável "*txtNode*";

Linhas 12 e 13: adição do nodo filho ao nodo principal;

Linhas de 15 a 17: adição do nodo principal ao documento XML em memória;

Linha 18: retorno do arquivo XML em formato texto.

Já na segunda iteração, foi implementada a regra de negócio sobre o estado (*state*), sendo que o trecho de código da Figura 75 mostra um exemplo de como foi aplicada essa regra de negócio.

```
1  switch (result[0]["state"].ToString())
2  {
3     case "AK":
4        strCity = "Juneau";
5        strZipCode = "99801";
6        break;
7     case "FL":
8        strCity = "Tallahassee";
9        strZipCode = "32301";
10       break;
11       ...
12    default:
13       strCity = "Albany";
14       strZipCode = "12201";
15       break;
16 }
```

Figura 75 - Exemplo de como a interface do estudo de caso implementou a regra de negócio sobre o estado

Dessa maneira, podemos descrever o código da Figura 75 da seguinte maneira:

Linha 1: clausula "*case*" com o estado (*state*);

Linhas de 3 a 6: se o estado for AK (Alaska), a cidade será Juneau e o CEP 99801;

Linhas de 7 a 10: se o estado for FL (Flórida), a cidade será Tallahassee e o CEP 32301;

Linhas de 12 a 15: caso não seja encontrado um estado correspondente, o padrão será NY (Nova York), a cidade será Albany e o CEP 12201.

- Após isso, o teste unitário é executado.

5.3 Segunda elicitação de requisitos

Uma segunda elicitação de requisitos foi necessária devido ao requisito de uma nova regra sobre o envio de dados XML, conforme podemos observar na Figura 76.

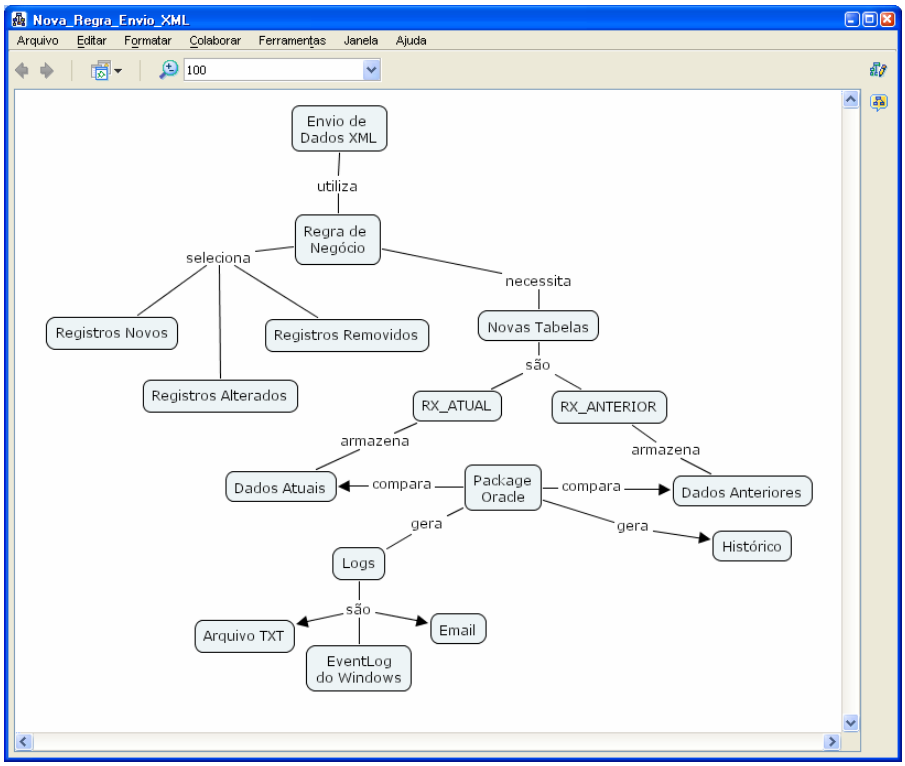


Figura 76 - Mapa conceitual com a nova regra de negócio sobre envio de dados XML

Podemos então perceber que esse requisito apresenta um ótimo exemplo de como a elicitação de requisitos é importante. Uma vez que um pequeno detalhe sobre o envio dos dados não foi percebido no início do projeto, ou seja, inicialmente foi entendido que todos os dados da tabela *EMPLOYEE* seriam enviados diariamente, porém percebeu-se depois que apenas os dados novos, alterados e removidos devem ser enviados. Ou seja,

se um registro de um empregado foi enviado hoje, não deverá ser enviado novamente amanhã, a não ser que o registro tenha sido alterado ou removido. No entanto, como estamos trabalhando com iterações pequenas de desenvolvimento, o impacto não foi muito prejudicial ao projeto.

5.3.1 Planejamento e decomposição

Após a segunda elicitação de requisitos, os desenvolvedores criaram uma nova estrutura de desmembramento de trabalho (EDT) para planejar as novas alterações necessárias, como apresentado na Figura 77.

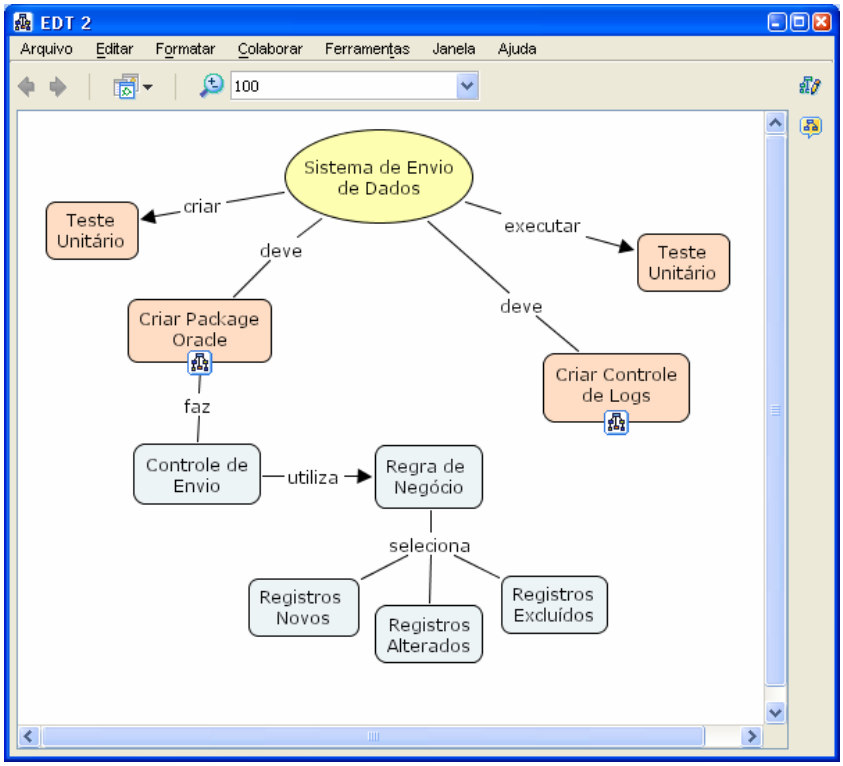


Figura 77 - Estrutura de desmembramento de trabalho criada no segundo release

Além disso, podemos observar que o requisito da Figura 76 foi divido em duas tarefas, sendo que o motivo dessa divisão foi o tempo de cento e quarenta horas estimado para a criação do pacote *Oracle* e o controle de *logs*. Assim, podemos ver a estimativa da criação do pacote *Oracle* na Figura 78 e a criação do controle de *logs* na Figura 79.

Um detalhe importante a ser destacado ainda são os nodos do tipo "Adorno" abaixo do nodo "Criar *Package Oracle*", cujo objetivo é acrescentar detalhes ao nodo do tipo *User Story*.

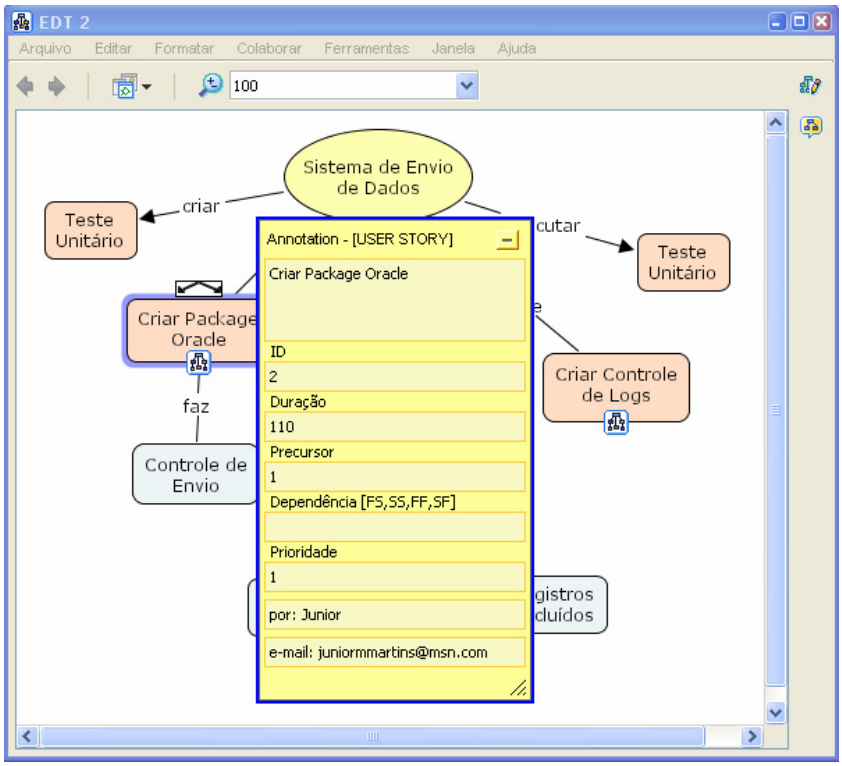


Figura 78 - Estimativa da tarefa "Criar Package Oracle"

Por sua vez, a tarefa “Criar Controle de *Logs*” foi estimada em trinta horas, como podemos ver na Figura 79.

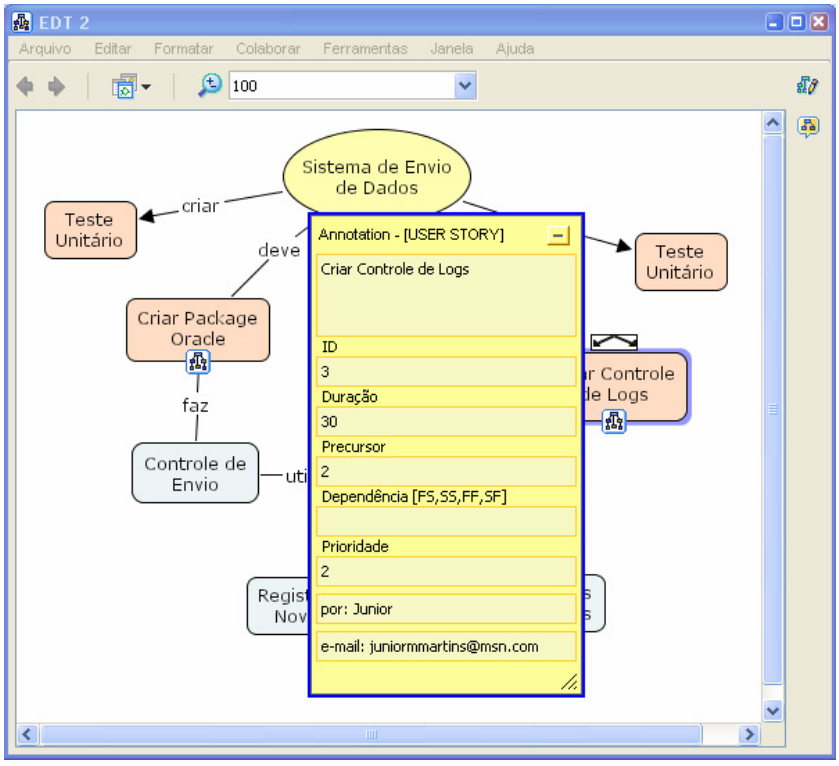
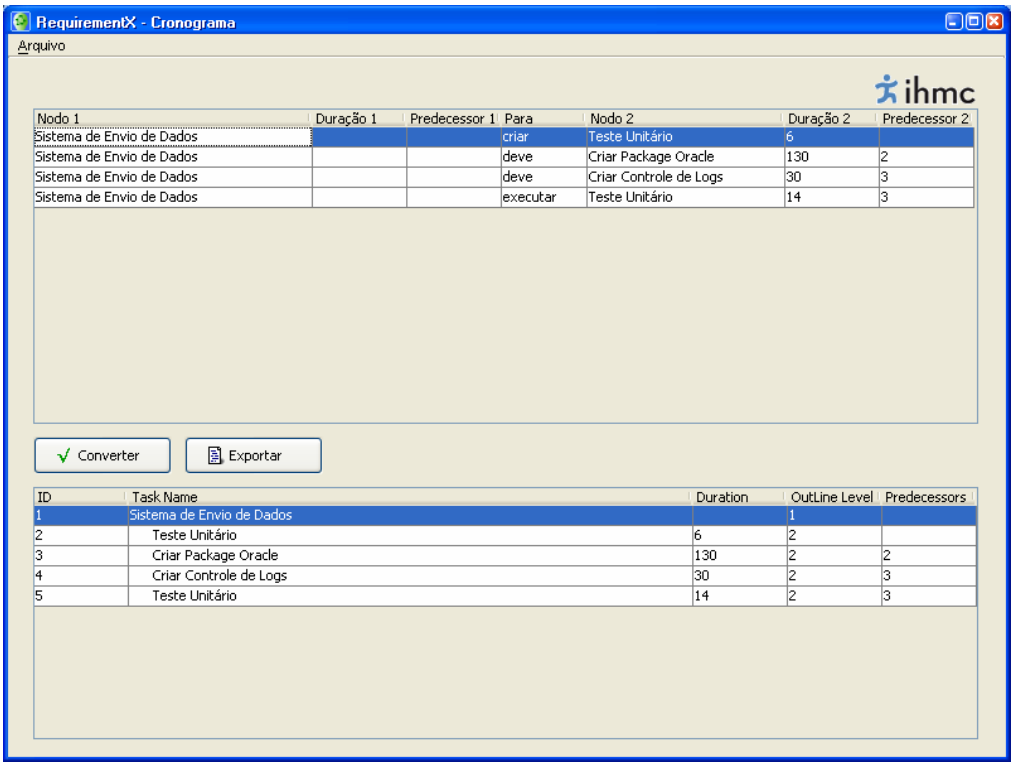


Figura 79 - Estimativa da tarefa "Criar Controle de Logs"

Após os desenvolvedores estimarem todas as tarefas (utilizando a interface exibida na Figura 78) e atribuírem suas prioridades (definidas pelos usuários), um novo cronograma foi gerado, como mostra a Figura 80.



Nodo 1	Duração 1	Predecessor 1	Para	Nodo 2	Duração 2	Predecessor 2
Sistema de Envio de Dados			criar	Teste Unitário	6	
Sistema de Envio de Dados			deve	Criar Package Oracle	130	2
Sistema de Envio de Dados			deve	Criar Controle de Logs	30	3
Sistema de Envio de Dados			executar	Teste Unitário	14	3

ID	Task Name	Duration	OutLine Level	Predecessors
1	Sistema de Envio de Dados		1	
2	Teste Unitário	6	2	
3	Criar Package Oracle	130	2	2
4	Criar Controle de Logs	30	2	3
5	Teste Unitário	14	2	3

Figura 80 - Geração de um novo cronograma com a interface de geração automática

Portanto, ao clicar no botão “Exportar”, um arquivo XML será criado e poderá ser importado pelo *software MS Project 2003*, como apresentado na Figura 81.

	Nome da tarefa	Duração	Início	Término	Predecessoras
1	**Sistema de Envio de Dados**	**22,5 dias**	**Sex 15/12/06**	**Ter 16/1/07**	
2	Teste Unitário	6 hrs	Sex 15/12/06	Sex 15/12/06	
3	Criar Package Oracle	130 hrs	Sex 15/12/06	Seg 8/1/07	2
4	Criar Controle de Logs	30 hrs	Ter 9/1/07	Sex 12/1/07	3
5	Teste Unitáriox	14 hrs	Sex 12/1/07	Ter 16/1/07	4

Figura 81 - Segundo cronograma gerado automaticamente pela ferramenta RequirementX

5.3.2 Implementação

Com a implementação da primeira iteração tivemos os seguintes resultados:

- Criação do teste unitário;
- O pacote *Oracle* foi criado, e é utilizado pela interface para fazer o controle dos registros novos, atualizados e removidos. Ou seja, se um registro já foi enviado, na próxima vez que a interface for executada esse registro não será enviado. Já se o registro for alterado ou removido, deverá ser enviado com o respectivo

status para que o *American Job Bank* atualize ou remova a informação de seus registros;

- A tabela *RX_RESULT* recebeu um novo campo chamado "*operation*" para definir o status do registro enviado, o qual deve receber os seguintes valores:

  N – se for um novo registro;

  U – se for um registro atualizado;

  R – se for um registro removido;

- As seguintes tabelas foram criadas, conforme podemos ver no diagrama de classes da Figura 82.

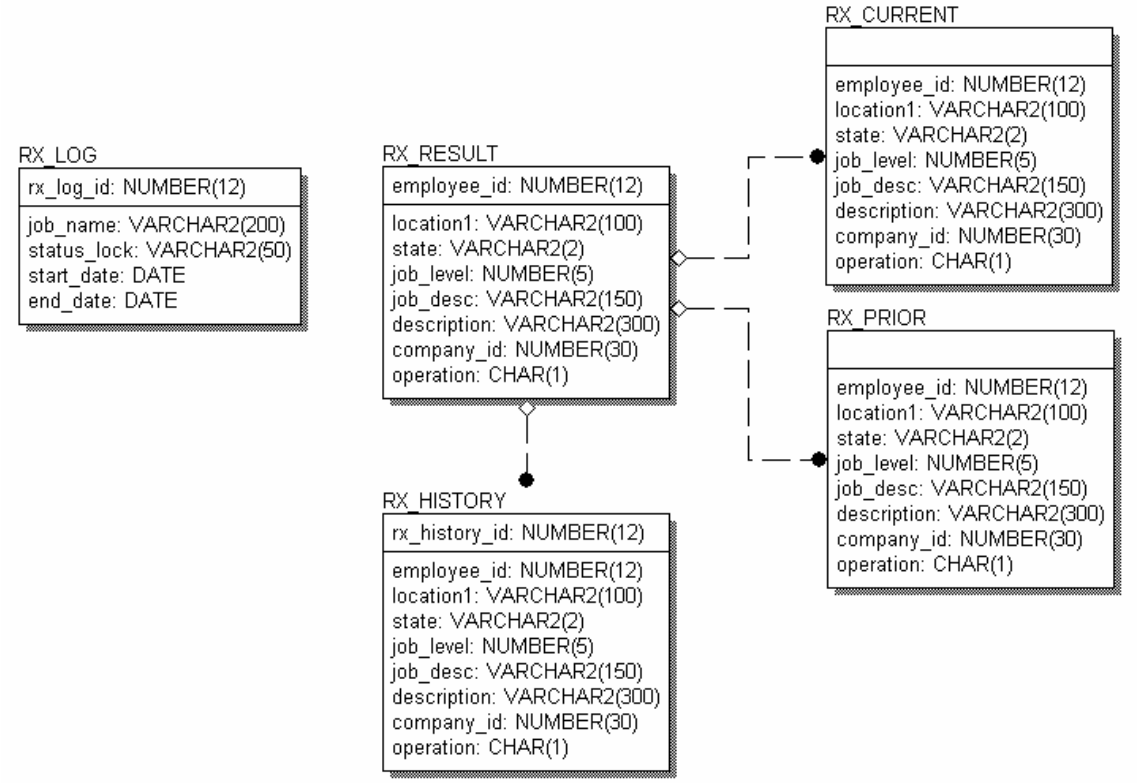


Figura 82 - Diagrama ER: tabelas utilizadas pelo pacote Oracle de envio de dados de empregados

5.4 Exportação dos requisitos para arquivos OWL

A *User Story* da Figura 63 será exportada, ou seja, o Mapa Conceitual sobre a definição das tecnologias utilizadas na interface de *software*, o que significa que o primeiro passo é modelar o Mapa Conceitual com a notação UML-MC, chegando a um modelo com uma semântica maior. Assim, o diagrama da Figura 83 mostra como o Mapa Conceitual da Figura 63 pode ser modelado em UML-MC.

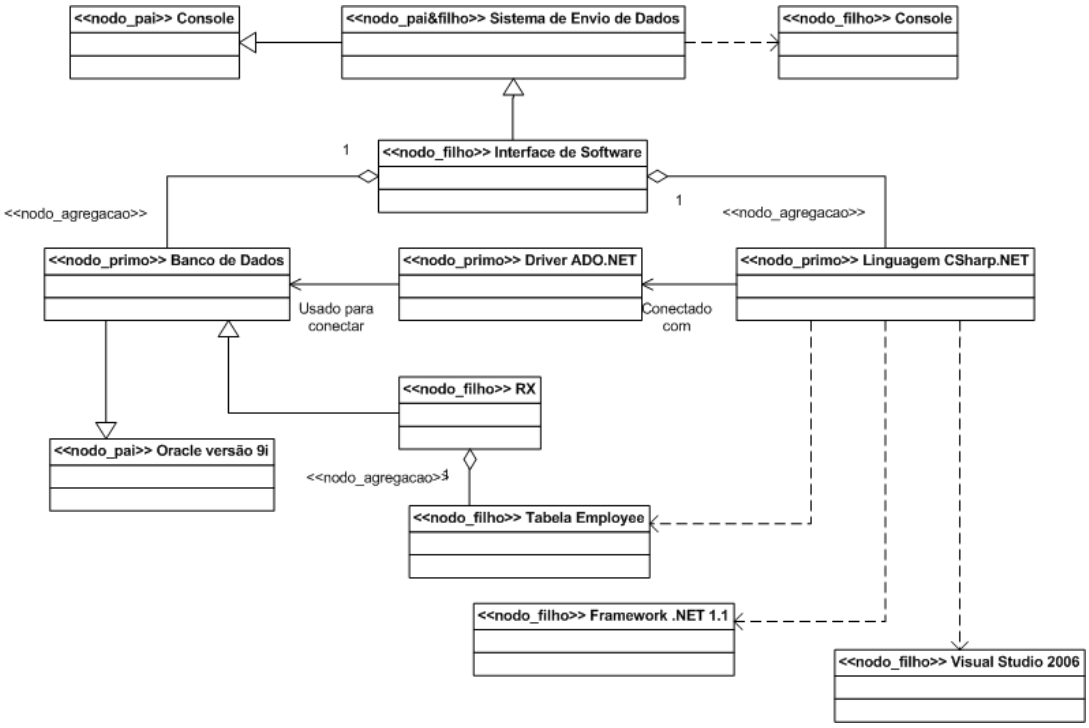


Figura 83 - Modelando o MC sobre a definição das tecnologias utilizadas

Podemos então observar que essa modelagem foi definida com a interface de interoperabilidade da ferramenta RequirementX, conforme a Figura 84.

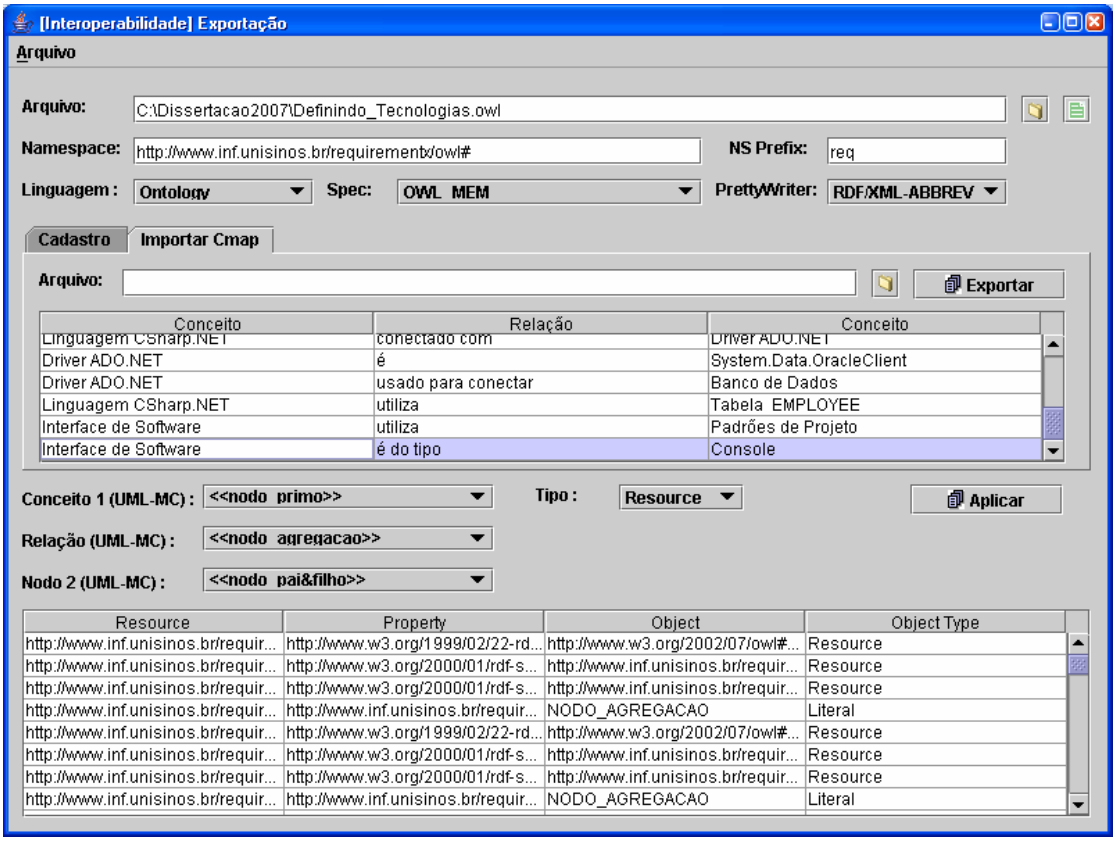


Figura 84 - Exemplo de exportação de Mapas Conceituais

Finalmente, ao clicar no botão “Exportar”, um arquivo OWL é gerado, sendo que podemos ver na Figura 85 um trecho desse arquivo gerado onde o seguinte relacionamento está sendo representado da seguinte maneira:

“Interface de Software -> é do tipo -> Console”

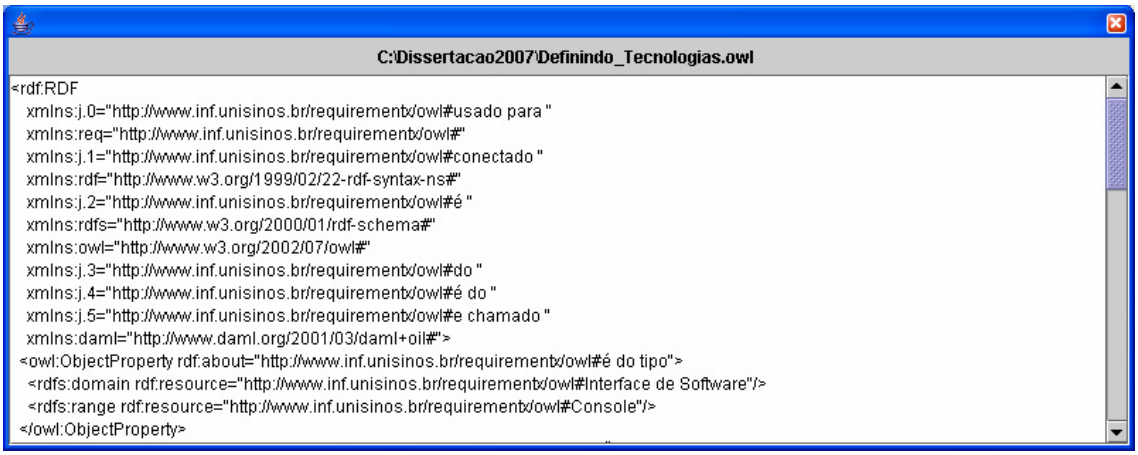


Figura 85 - Ontologia exportada para a User Story de definição de tecnologias

6 Conclusão

O objetivo deste trabalho foi usar Mapas Conceituais como meio de elicitação de requisitos de *software* na metodologia ágil *Extreme Programming* (XP), sendo que, para alcançar esse objetivo, foi criada uma ferramenta protótipo, chamada RequirementX, para permitir de forma colaborativa a descoberta, a gerência, a atualização e a representação do conhecimento (requisitos) oriundo das partes envolvidas na construção de *software* (usuários e desenvolvedores) para suporte à *Extreme Programming*, e que usa Mapas Conceituais para mapear o aprendizado de novos conceitos que norteiam o processo de construção de requisitos. Além do suporte à elicitação de requisitos, a ferramenta criada provê ainda suporte aos desenvolvedores no desmembramento da estrutura do trabalho e mecanismos de interoperabilidade com outras aplicações.

Sendo assim, para o desmembramento da estrutura do trabalho, a ferramenta RequirementX provê mecanismos que auxiliam na definição do tipo de conceito, cor, formato, armazenamento de informações específicas a tarefas, e geração de um cronograma. Portanto, visando à interoperabilidade, a ferramenta RequirementX provê um mecanismo de exportação de requisitos (em formato de Mapas Conceituais) para arquivos OWL com a utilização da API Jena 2. Entretanto, após alguns experimentos, verificou-se que a simples exportação de Mapas Conceituais para arquivos OWL não havia muito sentido, pois eles possuem a mesma estrutura de triplas de RDF. Então, foi incluída uma etapa antes da exportação, em que o Mapa Conceitual é modelado com a utilização da notação UML-MC para adicionar uma semântica aos dados exportados. RequirementX também oferece uma interoperabilidade com o *software MS Project*, através da exportação de cronogramas em formato XML.

Posteriormente, foi realizada uma comparação da ferramenta RequirementX com outras ferramentas que também provêem suporte à *Extreme Programming*, sendo que as ferramentas analisadas foram: PAM, XPSwiki, DotStories, MILOS, TUKAN, Web-Desktop e Storymanager. Concluímos assim que o uso de Mapas Conceituais na elicitação de requisitos de *software* em XP é a maior contribuição da ferramenta desenvolvida, pois eles têm sido empregados com muita eficiência em tarefas de captura de conhecimento, embora, apesar disso, nenhuma das ferramentas analisadas utiliza Mapas Conceituais. Outros pontos fortes da ferramenta RequirementX foram destacados, tais como: mecanismos de exportação, interoperabilidade, recursos visuais, ferramenta amigável, suporte à colaboração, e internacionalização.

Finalmente, um estudo de caso foi criado para o esclarecimento do uso da ferramenta proposta, cujo cenário foi baseado em uma regulamentação do governo americano que uma empresa deve seguir. Para isso, uma interface de *software* para a exportação de dados de empregados foi criada após três iterações, sendo que esse estudo de caso provou a viabilidade da proposta desta pesquisa. Porém, como a RequirementX é um protótipo, há alguns pontos que ainda podem ser melhorados na ferramenta, principalmente em relação ao controle de tarefas e à interoperabilidade.

Complementando, foi possível concluir também que a grande potencialidade da ferramenta RequirementX é a utilização de Mapas Conceituais, pois, quando conceitos e relações são escolhidos com cuidado, os mapas tornam-se ferramentas poderosas para observar a nuance dos significados. No entanto, o desenvolvimento do trabalho não pára por aqui, e, assim, a seguir serão abordados alguns pontos que necessitam de prosseguimento.

6.1 Trabalhos futuros

A seguir, são detalhados os principais pontos que devem ser tratados para que a solução proposta se torne uma ferramenta comercial:

- Realizar outros estudos de caso com empresas, buscando validar os conceitos estudados, o protótipo RequirementX desenvolvido nesta pesquisa, assim como também reunir sugestões de melhorias para a ferramenta;
- Desenvolver módulos de representação gráfica de informações gerenciais para a ferramenta RequirementX;
- Criar mecanismos inteligentes que auxiliem os usuários na elicitação de requisitos, baseando-se em conceitos de Inteligência Artificial;
- Criar um módulo de suporte a testes para as *User Stories* criadas com a ferramenta RequirementX;
- Adaptar a ferramenta RequirementX para auxiliar o processo de Engenharia de Requisitos de outras metodologias de desenvolvimento de *software*.

7 Bibliografia

[ALV 2007] ALVES, José Nelson A. JENA API. Disponível em: http://dme.uma.pt/jcardoso/Teaching/SemanticWeb/2005-06/Jena-Nelson/JenaAPI-Nelson%20Alves-2008299.doc, visitado em 10 de fevereiro de 2007.

[ANG 2006] ANGIONI, M.; CARBONI, D. e MELIS, M. XPSuite: Tracking and Managing XP projects in the IDE. http://delivery.acm.org/10.1145/1160000/1151440/p46-angioni.pdf?key1=1151440&key2=7000697611&coll=ACM&dl=ACM&CFID=15151515&CFTOKEN=6184618. Último acesso em 23 de dezembro de 2006.

[AST 2002] ASTELS, David; MILLER, Granville; NOVAK, Miroslav. Extreme Programming Guia Prático. Rio de Janeiro: Campus, 2002.

[AUS 1986] AUSUBEL, D.P., NOVAK, J.D. e HANESIAN, H. (1978). Educational psychology. New York: Holt, Rinehart and Winston. Publicado em português pela Editora Interamericana, Rio de Janeiro, 1980. Em espanhol por Editorial Trillas, México, 1981. Reimpresso em inglês por Werbel & Peck, New York, 1986.

[BAT 2005] BATISTA, Edinelson Aparecido; CARVALHO, Ariane M. B. R. Uma Taxonomia Facetada para Técnicas de Elicitação de Requisitos. http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER03/edinelson_batista.pdf. Último acesso em 15 de dezembro de 2005.

[BEB 2003] BEUTER, Elisângela Cristina; PINTO, Sérgio Crespo Coelho da Silva. Inserindo Mapas Conceituais como Ferramenta de Suporte para o Descobrimento de Requisitos. In: VIII SIMPÓSIO DE INFORMÁTICA E III AMOSTRA DE SOFTWARE ACADÊMICO, 2003, Uruguaiana. Revista Hífen. 2003. v.27, p.67-73.

[BEC 1999] BECK, Kent. Extreme Programming Explained: Embrace Change, Addison-Wesley Pub Co, 1999.

[BEC 2000] BECK, Kent; Fowler, Martin. Planning Extreme Programming, Addison-Wesley Pub Co, 2000.

[BEC 2004] BECKETT, D.; MCBRIDE, B. (2004). "RDF/XML Syntax Specification (Revised)", http://www.w3.org/TR/rdf-syntax-grammar/, acessado em maio de 2005.

[BER 1998] BERRY, Daniel M.; LAWRENCE, Brian. Requirements Engineering. IEEE SOFTWARE, Los Alamitos, p. 26-29, mar./abr. 1998.

[BEU 2003] BEUTER, Elisângela. Engenharia de Requisitos: Inserindo Mapas Conceituais como Ferramenta de Suporte para o Descobrimento dos Requisitos. Monografia submetida como requisito parcial à obtenção do grau do título de bacharel em Informática. Universidade do Vale do Rio dos Sinos – UNISINOS. São Leopoldo, 2003.

[BOO 1999] BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. The Unified Modeling Language Reference Manual. Califórnia: Addison-Wesley, 1999.

[BOO 2000] BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. Unified Modeling Language: Guia do Usuário. Rio de Janeiro: Campus, 2000.

[BOR 2005] BORTOLI, Lis Ângela de. Um Método de Trabalho para Auxiliar a Definição de requisitos.

http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana98/lis.html. Último acesso em 10 de dezembro de 2005.

[BRE 1999] BREITMAN, Karin Koogan; LEITE, Julio Cesar S. Do Prado and FINKELSTEIN, Anthony. The World's a Stage: A Survey on Requirements Engineering Using a Real-Life Case Study. J. Braz. Comp. Soc., July 1999, vol.6, no.1,p.13-37. ISSN. 0104-6500.

[BRO 1978] BROOKS Jr., Frederick P. The Mythical Man-Month, Addison-Wesley, 1978.

[CAB 2000] CABRAL, Anderson R. Yanzer. Estudo sobre Treinamento Baseado em Computador. Trabalho Individual II. Programa de Pós-Graduação em Ciência da Computação. Mestrado. PUC-RS. Maio, 2000.

[CAB 2002] CAÑAS, Alberto J. Algunas Ideas sobre la Educación y las Herramientas Computacionales Necesarias para Apoyar su Implementación. Disponível em: http://www.ihmc.us/users/acanas/Publications/IdeasEnEducacion/ACanas%20Ideas%20Educacion.htm. Acesso em 5 de maio de 2005

[CAC 2002] CAÑAS, Alberto J. Herramientas para construir y Compartir Modelos de Conocimiento Basados en Mapas Conceptuales. Disponível em: http://www.ihmc.us/users/acanas/Publications/RevistaInformaticaEducativa/HerramientasConsConRIE.htm. Acesso em 19 de maio de 2005.

[CAN 1993] CANÃS, Alberto J., Hill, Greg; Lott, James. Support for Constructing Knowledge Models in CmapTools, Technical Report IHMC CmapTools 93-02, 1993.

[CAN 2002] CANÃS, A. J.; M. CARVALHO; M. Arguedas. Mining the Web to Suggest Concepts During Concept Mapping: Preliminary Results, XIII Simpósio Brasileiro de Informática na Educação – SBIE – UNISINOS 2002, November 2002, Brasil.

[CAR 2001] CARVALHO, Ariadne M.B.R.; CHIOSSI, Thelma C.S; Introdução à Engenharia de Software. Campinas: Editora da UNICAMP, 2001.

[CAR 2005] CARVALHO, M. R.;CANÃS, A. J (2005). Mapas Conceituais e IA: Uma União Improvável? Pensacola, Florida: IHMC-University of West Florida, 2005.

[COF 2002] COFFEY, J. W.; HOFFMAN, R.; CANÃS, A.J.; FORD, K.M. A Concept Map-Based Knowledge Modeling Approach to Expert Knowledge Sharing, IKS 2002 – The IASTED International Conference on Information and Knowledge Sharing, November 2002, Virgin Islands.

[COF 2003] COFFEY, J. W.; CANÃS, A. J.; LEO: A Learning Environment Organizer to Support Computer-Mediated Instruction, Journal for Educational Technology System 31(3), 2003.

[CXO 2001] CxOne Point of View. Construx Software Builders, Inc. 2001. Acesso em: http://www.construx.com/Page.aspx?hid=801. Último acesso em 28 de dezembro de 2006.

[DER 2004] DERBENTSEVA, F. N.; Canãs A.J.; Concept Maps: A Theoretical Note on Concepts and the Need for Cyclic Concept Maps, Safayeni, submitted for publication, 2004.

[GON 2005] GONÇALVES, Paulo Roberto; BRITO, Parcilene Fernandes de. Manipulação de uma ontologia desenvolvida em OWL através da utilização da API JENA 2 Ontology. VI Encontro de Estudantes de Informática do Estado de Tocantins - ENCOINFO 2004, novembro de 2004.

[GRU 1999] GRUBBER, T. (1999) "What is an Ontology?". Disponível em: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html. Acesso em 01 de março de 2007.

[GUA 1998] GUARINO, N. (1998) "Formal Ontology and Information Systems", IN: First International Conference (FOIS), Trento, Itália. Anais: Trento: IOS Press, 1998.

[HER 2002] HERMANN, Kaindl; BRINKKEMPER, Sjaak; BUBENKO, Janis A. Jr; FARBEYD, Barbara; FARBEY, Sol; GREENSPAN, J.; HEITMEYERF, HEITMEYER, Constance L.; LEITE, Julio Cesar Sampaio do Prado; MEADH, Nancy R.; MYLOPOULOSI, John; SIDDIQIJ, Jawed; Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda, Requirements Engineering Journal: Vol. 7: 113-123, 2002. Springer-verlag London.

[HEW 2007] Hewlett-Packard (HP). Disponível em: http://www.hp.com. Acesso 15 de fevereiro de 2007.

[HOR 2000] HORROCKS, I.; FENSEL, D.; BROEKSTRA, J.; DECKER, S.; ERDMANN, M.; GOBLE, C.; HARMELEN, F.; KLEIN, M.; STAAB, S.; STUDER, R.; MOTTA, E. (2000). "The Ontology Inference Layer Oil", http://www.ontoknowledge.org/oil/TR/oil.long.html, maio de 2005.

[JAC 1995] JACKSON, Michael. Software requirements and specifications: a lexicon of practice, principles and prejudices. 1. ed. Massachusstes : Addison-Wesley, 1995. p. 228.

[JEN 2006] Jena 2 Ontology API. Disponível em: http://jena.sourceforge.net/ontology/. Acesso em 26 de dezembro de 2006.

[KIR 2001] KIRCHER, M.; JAIN, P.; CORSARO, A.; LEVINE, D. (2001), Distributed extreme programming, in 'XP2001 - eXtreme Programming and Flexible Processes in Software Engineering'.

[KOT 1998] KOTONYA, Gerald; SOMMERVILLE, Ian. Requirements engineering (Processes and techniques). 1. ed. England : J. Wiley & Sons, 1998. p. 282.

[LEF 2000] LEFFINGWELL, Dean; WIDRIG, Don. Managing Software Requirements: A Unified Approach - Addison-Wesley object technology series, Addison Wesley, 2000.

[LEI 2000] LEITE, Jair C. Análise e Especificação de Requisitos. http://www2.dem.inpe.br/ijar/EngSofAnalEspc.html. Último acesso em 12 de dezembro de 2005.

[LEI 2005] LEITE, Julio César Sampaio do Prado; LUCENA, Carlos José Pereira de. Organizando Processos de Requisitos. Disponível em: http://www.inf.puc-rio.br/~wer98/artigos/1.html. Acesso em 18 de maio de 2005.

[LEO 2002] LEONARDI, M.C.; LEITE, J.C.S.P. Using Business Rules in Extreme Requirements, CAISE 2002, LNCS 2348, pp. 420-435, 2002, Springer-Verlag.

[LUS 2003] LUSTOSA, P.A. (2003). "OWL e Protégé: estudo e aplicação de conceitos para exemplificação da definição da camada de esquema da Web Semântica em um

determinado domínio", Trabalho de Conclusão de Curso. Curso de Bacharel em Sistemas de Informação, Palmas.

[MAC 1996] MACAULAY, Linda A. Requirements engineering. 1. ed. Great Britain : Springer-Verlag London, 1996. p. 202.

[MAU 2002] MAURER,F. & MARTEL, S.(2002), Process Support for Distributed Extreme Programming.

[MOR 1982] MOREIRA, Marco Antonio. Aprendizagem significativa: a teoria de David Ausubel. 1. ed. - São Paulo: Moraes, 1982. 112 p.

[MOR 1987] MOREIRA, Marco Antonio. Mapas conceituais: instrumentos didaticos, de avaliacao e de analise de curriculo. 1. ed. São Paulo: Moraes, 1987. 83 p.

[NET 2002] NETO, José Ignácio Jaeger; BOCOLI, Fernanda Schmidt. Gerência de Projetos – Parte 2 e 3, Rio Grande do Sul - Gráfica UNISINOS, maio de 2002.

[NOV 1984] Novak, J. D., & Gowin, D. B. (1984). Learning how to learn. New York: Cambridge University Press.

[OMG 2007] Object Modeling Group. Disponível em: http://www.omg.org. Acesso 15 de fevereiro de 2007.

[ONT 2006] Ontology Definition Metamodel (Submitted by: IBM Sandpiper Software, Inc). Disponível em: http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf. Acesso em 15 de dezembro de 2006.

[OSH 2005] OSHIRO, Adriane K.; NOVELLI, Andréia D. P.; CASELI, Helena de M.; LUCENA, Percival de. “Extreme Programming, um novo modelo de processo para o desenvolvimento de software”. Disponível em < http://www.dc.ufscar.br/~rosangel/mds/Aula-09-XP/ArtigoXP.pdf>, visitado em 18 de dezembro de 2006.

[PIN 2003] PINNA, S.;LORRAI,P.;MARCHESI,M. & SERRA,N.(2003), Developing a Tool Supporting XP Process, in F. Maurer & D. Wells, eds, 'XP/Agile Universe 2003', pp. 150-160.

[PIN 2006] PINHEIRO, Manuele Kirsch; LIMA, José Valdeni de; BORGES, Marcos R.S. Awareness em Sistemas de Groupware. Disponível em: http://www-lsr.imag.fr/users/Manuele.Kirsch-Pinheiro/AwarenessV2.pdf. Acesso em 22 de dezembro de 2006.

[PRE 1995] PRESSMAN, Roger S. Engenharia de software. São Paulo: Makron, 1995.

[RDF 2006] An Introduction to RDF and the Jena RDF API. Disponível em: http://jena.sourceforge.net/tutorial/RDF_API/. Acesso em 10 de novembro de 2006.

[REE 2002] REES, Michael J. (2002), A Feasible User Story Tool for Agile Software Development?, in ' Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02)'.

[ROB 2003] ROBINSON, Genessa. UML-MC: Estendendo a Notação Gráfica da UML para Suportar Mapas Conceituais. Monografia submetida como requisito parcial a obtenção do grau do título de bacharel em Informática. Universidade do Vale do rio dos Sinos – UNISINOS. São Leopoldo, 2003.

[SCH 2001] SCHUMMER, T; SCHUMMER, J.(2001), Support for distributed teams in extreme programming, in G. Succi & M. Marchesi, eds, 'eXtreme Programming Examined', addison Wesley, pp. 355-377.

[SCH 2003] SCHNEIDER, Jean-Guy; JOHNSTON, Lorraine. Extreme Programming – Helpful or Harmful in Educating Undergraduates? Science Direct, 2003.

[SMI 2003] SMITH, M.K.;WELTY, C.;MCGUINNESS, D.L. (2003). "OWL Web Ontology Language Guide", http://www.w3.org/TR/2003/CR-owl-guide-20030818, maio de 2005.

[SOM 1997] SOMMERVILLE, Ian; SAWYER, Pete. Requirements engineering (A good practice guide). 1. ed. England : J. Wiley & Sons, 1997. p. 391.

[SUZ 1999] R., Suzanne. Mastering The Requirements Process. Harlow: Addison-Wesley, 1999.

[SWE 2005] http://www.w3.org/2001/sw/. Visitado em 22 de maio de 2005.

[VAR 1998] VARGAS, Ricardo Viana. Gerenciamento de projetos com o MS Project 98: estratégia, planejamento e controle – Rio de Janeiro: Brasport, 1998.

[VER 2000] VERZUH, Eric, MBA Compacto – Gestão de Projetos. Rio de Janeiro: Campus, 2000.

[VER 2001] VERZULLI, J. Using the Jena API to Process RDF. O'Reilly, 2001. Disponível em: http://www.xml.com/pub/a/2001/05/23/jena.html. Acesso em 26 de maio de 2004.

[WIL 2003] WILLIAMS, Malcolm Maxwell. Distributed Extreme Programming: Extending the Frontier of the Extreme Programming Software Engineering Process. A thesis submitted in partial fulfilment of the requirements for the Degree of Master os Science. University of Canterbury, 2003.

[XIS 2005] http://www.xispe.com.br, visitado em 15 de maio de 2005.

[ZAN 1998] ZANLORENCI, Edna Pacheco; BURNETT, Robert Carlisle. Engenharia de Requisitos (RE - Requirements Engineering): Conceitos e Fundamentos. Disponível em: http://www.pr.gov.br/batebyte/edicoes/1998/bb77/engenharia.htm. Acesso em 01 de março de 2007.

[ZAV 1997] ZAVE, Pamela; JACKSON, Michael. Four dark corners of requirements engineering. Transactions on software engineering and methodology, USA, v. 6, n.1, p. 1-30, Jan. 1997.