UNIVERSIDADE ESTADUAL DE MARINGÁ
PROGRAMA DE INICIAÇÃO CIENTÍFICA - PIC
DEPARTAMENTO DE INFORMÁTICA
ORIENTADORA: Prof^{a}. Dr^{a}. Elisa Hatsue Moriya Huzita
ACADÊMICO: Marco Aurélio Graciotto Silva

Uma Ferramenta para Apoiar a Definição de Requisitos no Desenvolvimento de Software Distribuído

Maringá - PR, agosto de 2001.

UNIVERSIDADE ESTADUAL DE MARINGÁ
PROGRAMA DE INICIAÇÃO CIENTÍFICA - PIC
DEPARTAMENTO DE INFORMÁTICA
ORIENTADORA: Prof^{a}. Dr^{a}. Elisa Hatsue Moriya Huzita
ACADÊMICO: Marco Aurélio Graciotto Silva

Uma Ferramenta para Apoiar a Definição de Requisitos no Desenvolvimento de Software Distribuído

Relatório final de projeto
de iniciação científica

Maringá - PR, agosto de 2001.

Resumo

As recentes tendências do mercado têm mostrado que a complexidade do software continuará a crescer drasticamente nas próximas décadas. Aliada a isto, a globalização acaba por envolver organizações de diferentes portes, com políticas peculiares de tomadas de decisão. O volume de dados a ser utilizado cresce ao mesmo tempo que temos uma descentralização deste. Neste novo panorama, sistemas isolados, monolíticos, são uma solução pouco eficaz, dando lugar aos sistemas distribuídos. O advento de sistemas distribuídos leva a aplicações mais complexas, implicando que desenvolver software usando métodos tradicionais torna-se ineficiente. A redução desta complexidade pode ser obtida através da decomposição, estruturação e delegação de tarefas, empregando para isto metodologias de desenvolvimento adequada. A criação de ferramentas que dêem suporte a tais metodologias é desejável. O objetivo deste projeto é o desenvolvimento de uma ferramenta que auxiliará na definição de requisitos de sistemas distribuídos, a ser utilizada na Metodologia de Desenvolvimento Baseado em Objetos Distribuídos Inteligentes (MDSODI). Foram estudadas várias abordagens que podem ser utilizadas para a definição de requisitos: pontos de vista; uso de padrões na construção de cenários; padrões de requisitos e utilização de modelos para descrição, qualificação, análise e validação de requisitos. Destas, duas foram escolhidas: ponto de vistas por possibilitar a rastreabilidade dos casos de uso, ser facilmente utilizados nos diversos processos de engenharia de software e de fácil aplicabilidade; e utilização de modelos para descrição, qualificação, análise e validação de requisito, que apóia a resolução dos conflitos das visões sobre o sistema, determinando critérios para esta tarefa. O projeto desta ferramenta está documentado em UML e implementado em Java, utilizando o ORBacus como middleware CORBA e o banco de dados PostgreSQL para o depósito.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

As recentes tendências do mercado têm mostrado que a complexidade do software continuará a crescer drasticamente nas próximas décadas. Aliada a isto, a crescente globalização acaba por envolver diferentes organizações em direntes lugares com opolíticas peculiares de tomadas de decisão, grandes organizações utilizam-se de um grande volume de dados que, normalmente, encontram-se dispersos em diferentes lugares. Deste modo, os produtos de software isolados estão caindo em desuso à medida que são cada vez mais disseminadas a Internet e as Intranets.

O advento de sistemas distribuídos heterogêneos leva a aplicações mais complexas implicando que desenvolver software usando métodos tradicionais tem se tornado cada vez mais inadequado. A redução desta complexidade pode ser obtida através da decomposição, estrututação e delegação de tarefas, empregando para isto uma metodologia de desenvolvimento adequada. Portanto, os modernos engenheiros de software deverão adequar os sistemas que projetam a esta nova realidade. Tais fatos induzem a uma procura por técnicas alternativas que possam ser utilizadas no desenvolvimento de software como, por exemplo, a utilização de agentes inteligentes [KNAPIK, 1998; WOOLDRIDGE, 1999].

É importante notar ainda que a distribuição pode se referir não apenas ao processamento mas, também, ao seu desenvolvimento. Este fato nos leva a necessidade de adotar novas tecnologias e condutas no processo automatizado para o desenvolvimento deste tipo de software. Em [HUZITA, 1995], são analisadas várias ferramentas que dão suporte ao desenvolvimento de software paralelo, tais como PO, GRASPIN, VISTA, PROOF, TRAPPER, PARSE, mas tais ferramentas trabalham os aspectos de programação, não apresentando preocupações quanto ao processo de engenharia de software, à exceção de PROOF e PARSE. Encontra-se na literatura referências a alguns poucos outros ambientes de desenvolvimento de software: ONIX [SATO, 1994, p. 167–183], ABACO [SOUZA, 1998, p. 205–220], PROSOFT [SCHELEBBE, 1995], sendo somente estes dois últimos destinados ao desenvolvimento de software distribuído. Assim, encontra-se em andamento um projeto para definir uma metodologia para desenvolvimento de software baseado em objetos distribuídos inteligente (MDSODI), oferecendo recursos de reusabilidade de componentes.

Um aspecto fundamental no processo de engenharia, abordado neste projeto, é a definição de requisitos. Utilizando técnicas tais como pontos de vista, qualificação automática de requisitos, identificação de padrões de reuso, aliada com extensões de linguagens de modelagem, empregando uma ferramenta para automatizar a sua aplicação no processo, problemas típicos deste etapas são abordados.

2 Objetivos

O objetivo deste projeto é desenvolver um protótipo de uma ferramenta para apoiar a definição de requisitos para projeto de software distribuído. Os objetivos específicos são:

- Estudo de processo de desenvolvimento de software;
- Estudo da MDSODI - Metodologia de desenvolvimento baseada em objetos distribuídos inteligentes;
- Definição da arquitetura do protótipo;
- Especificação da interface do protótipo da ferramenta;

- Implementação de um protótipo da ferramenta para dar suporte à especificação de requisitos;
- Avaliação do protótipo utilizando-o em estudos de caso.

3 Materiais e métodos

3.1 Repositórios

Repositórios são aplicações que possibilitam o armazenamento de dados utilizados e gerados no processo de engenharia. Eles permitem um acesso transparente às informações nele armazenadas, possui mecanismos de controle de versão, aplica controle de acesso aos dados. Além disto, geralmente os repositórios armazenam algum tipo de metadado que possibilita a extração de conhecimento do conteúdo nele armazenado. Foram estudados alguns depósitos: kMail, WebCADET, o desenvolvido pelo projeto NIST Design Repository, Unisys UREP. Todos eles são repositórios, cada qual com uma ou outra característica específica quanto ao armazenamento de dados, mas sempre com uma metodologia para tratamento do conhecimento diferente.

3.1.1 kMail

Praticamente todas as empresas possuem uma base de conhecimento ampla porém, muitas vezes, subutilizadas. Muito provavelmente, isto se deve a falhas em alguma das atividades de gerenciamento desta base: aquisição de novos dados, a organização destes ou sua distribuição. Em pesquisa realizada por Daniel O'Leary [O'LEARY, 1998], são apontadas algumas razões desta subutilização:

- As atividades realizadas no banco devem ser orientadas a ações;
- As bases de conhecimentos devem possuir mecanismos eficientes para facilitar a incorporação de novos dados, geração de conhecimento e sua consequente atualização;
- O conhecimento não deve substituir a criatividade e sim estimulá-la e guiá-la.

A partir do momento em que a base começa a fazer diferença nos processos da empresa, aumentando seus lucros, o sistema começa a ganhar respeito, valor e, conseqüentemente, mais investimento.

O kMail é uma ferramenta (dividida em duas partes: uma cliente e outra servidor) que torna disponível o conhecimento organizacional de acordo com as ações das pessoas, usando para isto uma base de conhecimento acessível pela Internet (através de URLs), uma base de metaconhecimento e a ferramenta que, utilizando dessas bases, fornece a informação mais apropriada de acordo com as ações em execução, resolvendo os pontos criticados por Daniel quanto a subutilização das bases de conhecimentos da empresa.

Um importante aspecto do kMail é que as ações consistem em trocas de emails. A escolha deste meio de comunicação deve-se ao fato de que, muito provavelmente, este serviço já esteja implementado na empresa, o que torna a curva de aprendizado para este novo sistema muito mais suave. Pesquisas também provam que o email é utilizado, na maioria dos casos, para

requisitar ou responder algo, ou seja, exatamento no momento em que os dados que temos na base de conhecimento organizacional fazem-se necessários.

Além do serviço de email, o kMail possui dois outros requisitos: os conhecimentos da base, acessiveis pela Internet, e um sistema de metaconhecimento para facilitar o acesso às informações da base. O kMail não trata da organização do conhecimento, este é o seu requisito principal. Já o sistema de metaconhecimento é uma parte fundamental do kMail. Armazenando dados dos emails enviados (remetente, destinatário, assunto, data de envio) e relacionando estes dados com os perfis dos usuários (cargo, experiência, etc), determina-se o contexto situacional que vai possibilitar a criação de visões da base de conhecimento. Destas escolhe-se a mais relevante para ser utilizada.

Detalhando mais o funcionamento do kMail, cita-se um pseudo-exemplo de como seria a utilização do sistema:

1. Envia-se um email para o setor de marketing da empresa a respeito de um novo produto: a calça jeans vermelha XYZ. Para criar este email, utiliza-se um cliente kMail.
2. O cliente kMail faz uma análise do email, identificando atributos importantes do email necessários à criação do contexto. Em seguida, envia-se este email para o servidor kMail.
3. O servidor kMail, com os dados recém-recebidos, os perfis de usuários e o meta-conhecimento já existente sobre a base de dados, cria várias visões (pessoal, supervisional, relacionada ao projeto, relacionada ao cargo).
4. Esses dados são repassados ao cliente kMail. Agora, o autor do email deve inserir links que considere relevantes, valida os já existentes e, enfim, confirma o repasse do email para os destinatários.
5. Os destinatários recebem o email em programa de email comum (que aceite trabalhar com emails no formato HTML). Esses emails vem com os links selecionados pelo autor do email, a visão que ele selecionou.

Com isto, consegue-se tirar proveito da base de conhecimento de maneira transparente e rápida. A validação dos links (pelo autor) e os acessos aos links (pelos destinatários) permitem determinar a importância dos dados da base, que dados não são utilizados, permitindo assim a obtenção de métricas para melhorar a organização da base e melhorar a criação de visão e o aprimoramento do mecanismo de meta-conhecimento, fazendo-o considerar estes diversos graus de utilização dos dados.

3.1.2 WebCATET

O WebCADET consiste numa ferramenta para suporte a decisão baseada na Web, permitindo assim seu uso por várias pessoas em diferentes locais do mundo, ou seja, de maneira distribuída. Neste projeto adotou-se o paradigma de "IA como texto", que possibilita ver o conhecimento da base na forma de texto, legível por seres humanos (obedecendo regras de gramática e vocabulário adequados para isso), garantindo assim transparência ao usuários do sistema.

O principal do WebCADET é a sua representação do conhecimento, estruturada de acordo com o grau de detalhamento, começando no nível mais baixo. Por exemplo, o sistema começa com a opção de escolher o setor ao qual o produto a ser projetado melhor se adequa, posteriormente temos a escolha do tipo de produto, em seguida características específicas do mesmo.

Este seria o penútimo nível da hierarquia do sistema que interagiria com as regras que temos sobre os produtos (que seria o último nível).

As regras, que são a base fundamental do suporte à decisão, juntamente com o mecanismo de inferência, são estruturadas da seguinte forma:

- **rule_id**: Identificador único da regra em todo o sistema.
- **name**: Nome da regra.
- **preconditions**: Pré-condições que devem ser atendidas.
- **conditions**: Permite qualificar os atributos do produto sendo proposto. Basicamente são regras condicionais como, por exemplo, "if button_number_size gt 3 then 3 else 0".
- **scale**: Fator normalizador.
- **history**: Histórico da regra: quando foi criada e por quem, quando foi alterada, o que foi alterado, etc.
- **keywords**: Palavras chaves para ajudar a identificar o contexto da regra.

O sistema tem três modos de uso: avaliação de projetos, consulta à base de conhecimento e adição de novos conhecimentos. Em avaliação de projetos, o projetista detalha os dados de seu projeto e o sistema, aplicando as regras cabíveis, verifica quão bem sucedido seria o produto, atribuindo uma nota ao produto. Pode-se também verificar o porquê de cada nota, com o sistema retornando um texto com base nas regras (3.1.2aplicadas (aqui temos o emprego da "IA como texto"). Outro modo de operação é o de consulta. Neste apenas percorre-se a base de conhecimento, sendo útil para verificar os dados que os sistema já tem armazenado. Enfim, temos o modo que permite a inserção de conhecimento, no qual podemos criar novos produtos, adicionar regras a eles (ou então a produtos já existentes).

```
rule_id phone_easy_to_dial_1
name easy_to_dial
precondition  product_type phone
conditions
[
if button_shape iaiof easy_dial_button_shape then 4 else 0.
 if button_length gt 5 and button_length lt 18 then 5 else 0.
 if button_width gt 5 and button_width lt 18 then 5 else 0.
 if button_configuration iaiof easy_dial_button_configuration then 3 else 0.
 if button_spacing gt 6 and button_spacing lt 20 then 5 else 0.
 if button_material has_aspect easy_dial_button_material then 2 else 0.
 if button_number_size gt 3 and button_number_size lt 16 then 3 else0.
 ]
scale  27
history
[amendent(author(`Paul Rodgers`,`EDC, Department of Engineering,
Cambridge University, UK`,
`pr2@eng.cam.ac.uk`),creationdate(30,`April,`1993),
`This attribute was determined to be relevant for this type of product as
 the result of field work undertaken by the author`)]
keywords [`generic`].
```

Figura 1: Exemplo de regra do Webcadet.

Alguns aspectos do WebCadet foram criticados em testes de usabilidade: impossibilidade de salvar sessões no sistema (o que obrigaria a criação de produtos em questões de minutos,

sem interrupções) e navegabilidade (um tanto quanto confusa). Mas a base do sistema em si, o mecanismo de inferência e a conseqüente aplicação das regras, possibitando a rápida avaliação de projetos, é funcional, inclusive atraindo a atenção de algumas universidades e até empresas interessadas em testar o sistema, conforme relatado no artigo estudado.

3.1.3 Repositório de Projetos do NIST

O Repositório de Projetos NIST objetiva a criação de uma linguagem de modelamento de projeto, interfaces para manipular os artefatos do repositório, identificação de taxonomias de funções e fluxos associados às mesmas e a criação de um protótipo de repositório de projetos. Por ser financiado por empresas, o projeto é bem prático, funcional e objetivo.

A representação dos artefatos do depósito é feita em torno de três parâmetros: forma, função e comportamento. Adotar-se uma linguagem específica para descrevê-los, representando os artefatos através de um conjunto de objetos e seus relacionamentos, tentando ao máximo a utilização de termos únicos entre as descrições, seguindo uma estrutura definida. Outra importante característica é a criação de taxonomias para funções e fluxos. Essas taxonomias permitem a modelagem de funções para uma grande variedade de artefatos.

3.1.4 Osirix

O Osirix é um sistema gerenciador de conhecimento que, utilizando documentos XML validados com DTDs, possibilita a procura por informações de maneira ágil e eficaz.

O processo de criação das DTDs e agregamento de novos documentos XML no repositório pode ser assim descrito:

1. O primeiro passo para a construção do depósito é a definição da ontologia para uma memória corporacional (base de conhecimento da empresa).. Isso é feito com a linguagem CML (CommonKADS Conceptual Modeling Language).
2. Define-se então um modelo padrão para os documentos XML que serão armazenados no repositório, criando a *core DTD*.
3. Aqui o Osirix faz sua primeira aparição, transformando a ontologia definida em CML em um DTD e integrando-a com a *core DTD*.
4. Agora temos a fase de alimentar o depósito. O autor de documentos deve enviá-los em um formato válido, de acordo com uma DTD específica definida no repositório.
5. O repositório valida este documento XML. Caso esteja correto, armazena-o.

Enfim, define-se o mecanismo de busca do repositório. Ele adota um modelo híbrido, fazendo primeiramente uma busca tradicional e, depois, uma busca utilizando-se da ontologia dos documentos encontrados. O primeiro método de busca usa mecanismos semelhantes aos do Google, Altavista e afins, sendo extremamente rápido porém muito limitado, suportando apenas restrições do tipo AND, OR, NOT. A segunda busca é feita observando os atributos dos documentos XML encontrados e as palavras chaves da buscas. Este segundo método é o mais importante de todos, por isso cabe a ele um maior detalhamento:

1. Identificam-se os elementos XML dos documentos XML encontrados na busca anterior.

2. Procura-se a presença semântica de uma palavra chave (conceito ou propriedade) requerida na pesquisa. Este presença semântica significa a presença de uma palavra chave como um *tag* na descrição ontológica do documento.
3. Verifica-se se a tag encontrada possui o valor requisitado na pesquisa. Caso positivo, cria-se uma página HTML a partir do documento XML encontrado (utilizando para isso o XSL) e envia o resultado como resposta da pesquisa.

3.2 CORBA

Um sistema distribuído consiste em vários componentes, localizados em computadores ligados por uma rede, que se comunicam e coordenam através de passagem de mensagens. Disto deriva-se várias características: concorrência de componentes, ausência de uma hora global (dificultando a sincronização) e falhas dos componentes. Um sistema deste tipo precisa atender características tais como segurança, independência de escala, abertura, heterogeniedade dos componentes, transparência. Construir um sistema com tamanha capacidade é, obviamente, uma tarefa difícil. A fim de facilitar a realização de tais sistemas, várias tecnologias foram desenvolvidas: CORBA, RMI, Jini, DCOM, SOAP. Destas, a única solução aberta, madura, com implementações gratuitas, capaz de funcionar em ambientes heterogêneos, é o CORBA.

O Common Object Request Broker Arquitecture (CORBA) foi criado pela Object Management Group (OMG), uma organização internacional com mais de 800 membros (empresas, engenheiros de software e usuários). Seu objetivo é servir como plataforma para a construção de sistemas distribuídos baseados em objetos e, ao mesmo tempo, possibilitar a utilização de software legado no sistema. Sua arquitetura compõe-se de quatro componentes:

- Object Request Broker (ORB): permite a comunicação transparente entre os objetos no sistema disbribuído
- Object Service: provêem serviços de baixo nível, tal como localização, persistência.
- Common Facility: são ferramentas que permitem a construção de sistemas num domínio específico.
- Application Object: são as aplicações existentes no sistema distribuído, geralmente feitas com auxílio dos serviços e facilidades CORBA.

O ORB provê meios para que requisições sejam feitas pelos objetos de maneira transparente, provendo assim interoperabilidade entre aplicações em diferentes máquinas em sistemas distribuídos heterogêneos.

Para fazer uma requisição, um cliente pode utilizar-se da invocação dinâmica ou de subs IDL. O ORB, por sua vez, precisa repassar a requisição para a implementação do objeto requerido apropriadamente. Isso pode ser feito utilizando-se esqueletos IDL ou através de esqueletos dinâmicos. Essa flexibilidade se deve aos mecanismos de definição de interfaces do CORBA: interfaces estáticas definidas com a linguagem IDL (Interface Definition Language) e repositórios de interfaces, no qual as interfaces são definidas em tempo de execução. A figura 3.2 retrata toda esta estratégia.

Sua complexidade é proporcional à sua importância na arquitetura, sendo necessário o estabelecimento de interfaces padronizadas quando visto de fora e proprietárias quanto a arquitetura interna de cada ORB. Em 3.2 é possível observar as diversas interfaces do CORBA. As

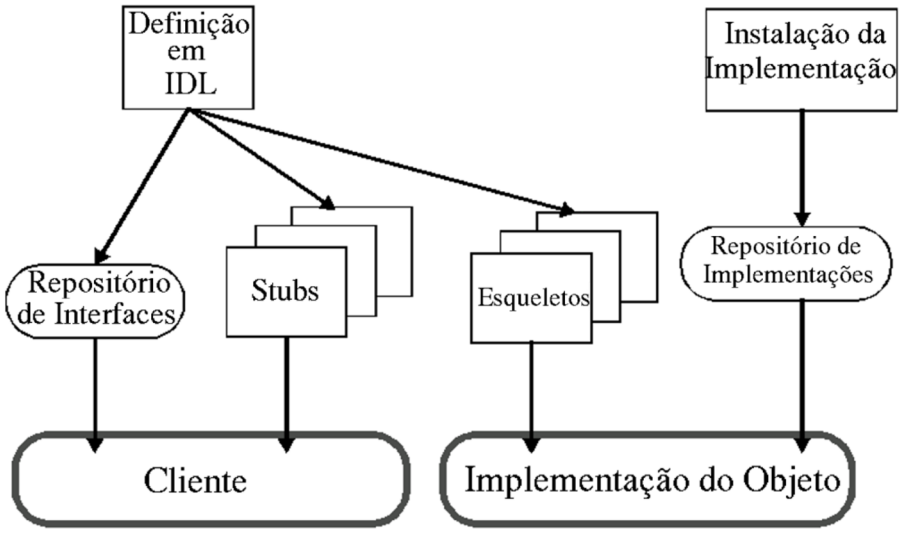


Figura 2: Mecanismos para requisição e acionamento de objetos

interfaces preenchidas com preto são comuns a todo ORB enquanto que as cinza claro são proprietárias, variando de implementação para implementação.

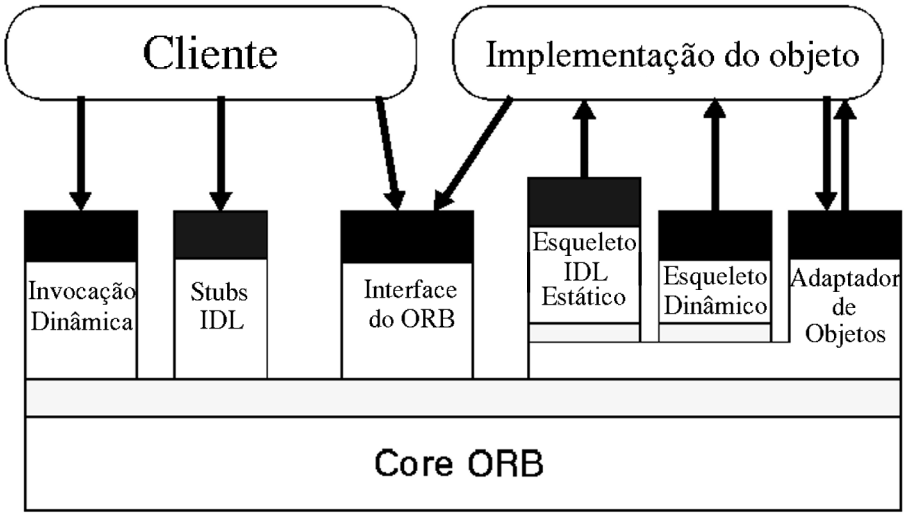


Figura 3: Interfaces do Object Request Broker

3.3 Técnicas aplicáveis no processo de engenharia de requisitos

3.3.1 Uso de padrões na construção de cenários

Uma alternativa aos métodos que empregam casos de uso é a utilização de cenários. Estes diferem do primeiro por conterem mais informações, serem tipados, ou seja, emprega-se tipo de atores ao invés de atores reais do domínio da aplicação. Apesar destes acréscimos, mantém-se a acessibilidade deste método quando se comparado ao de casos de uso.

Uma técnica efetiva para construir tais cenários é através do vocabulário do Universo de Discurso, ou seja, as palavras mais utilizadas quanto a aplicação em questão. Este trabalho utiliza-se de uma estrutura denominada LEL (Léxico extendido da linguagem) que permite registrar tal vocabulário e sua semântica, deixando para uma etapa posterior a compreensão do problema. Cada símbolo descoberto é identificado por uma palavra ou frase relevante no domínio da aplicação, utilizando-se linguagem natural para isso, facilitando a comunicação com o stakeholder.

A identificação de padrões nestes cenários permite a reutilização de soluções já existente em problemas similares. No entanto, é necessária uma técnica eficiente para identificar esses padrões. Este é o enfoque do artigo estudado, detalhado no texto a seguir.

Seguindo a estratégia "divisão e conquista", os cenários são tratados como compostos de subcenários ou diversos episódios . Para cada um deste são considerados os seguintes aspectos:

- Número de atores envolvidos;
- Atores requerem resposta ou não;
- A resposta deve ser imediata ou deferida;
- O papel desempenhado pelo ator.

De todos os critérios acima, o mais importante, representativo, é o do papel desempenhado pelo ator, sua participação nos sub-cenários ou episódios. Baseando-se nisto, propõe-se a seguinte classificação para os episódios:

- **p** (produção): um único ator, de maneira autonôma, realiza uma troca com o macrosistema.
- **s** (serviço): um dos atores adquire o papel de ator ativo e realiza uma ação em benefício de um ou mais atores passivos.
- **c** (colaboração): dois ou mais atores realizam uma ação que requer a participação de todos eles, produzindo um efeito global no sistema.
- **d** (demanda): um dos atores desempenha um papel ativo e um ou mais são passivos, sendo que as ações do ator ativo exigem, implicitamente, a resposta dos atores passivos.
- **r** (resposta): um ator, que fora passivo em um episódio do tipo **d**, assume o papel ativo e atende o pedido (responde a requisição do ator ativo no episódio **d**).
- **i** (interação): são episódios que reunem as propriedades dos episódios de resposta (**r**) e demanda (**d**), atendendo um pedido prévio e gerando um novo.

Definidas as classificações dos episódos e sub-cenários, pode-se construir inúmeras situações com características bem definidas. Por exemplo, uma seqüência de episódios que começa com um do tipo **d** e continua com vários do tipo **i** implica na existência de dois ou mais atores realizando uma atividade interativa na qual uma ação de um ator provoca uma ação de outro ator e assim por diante. A esta seqüência de episódios da-se a classificação de Negociação. Porém, a classificação das situações não se restringe somente aos episódios, todo e qualquer elemento que pertença ou influa no cenário pode ser considerado.

No estudo realizado, vários tipos de situações foram definidos de acordo com estes critérios:

- **Produção**: realização de uma atividade produtiva que provocará um efeito sobre o macrosistema;
- **Serviço**: prestação de um serviço que é necessário para um dos atores;
- **Colaboração**: associação de vários atores para realizar uma atividade cooperativa com um objetivo comum;
- **Negociação inconclusiva**: iniciação de uma atividade que requer uma sequência coordenada de ações por parte dos atores, necessitando de outra situação para concluir a negociação;
- **Negociação inconclusiva com disparo de cenários**: iniciação de uma atividade que requer uma sequência coordenada de ações por parte dos atores, criando a necessidade de várias outras situações;
- **Final de negociação**: sequência coordenada de ações por parte dos atores que finaliza uma atividade iniciada em outro cenário;
- **Etapa de negociação**: sequência coordenada de ações por parte dos atores que continua uma atividade de uma situação anterior e cuja finalização é inconclusiva;
- **Etapa de negociação com disparo de cenários**: sequência coordenada de ações por parte dos atores que continua uma atividade de uma situação anterior e cuja finalização resultará em várias outras situações;
- **Negociação terminada**: fim de uma atividade que requer uma sequência coordenada de ações por parte dos atores.

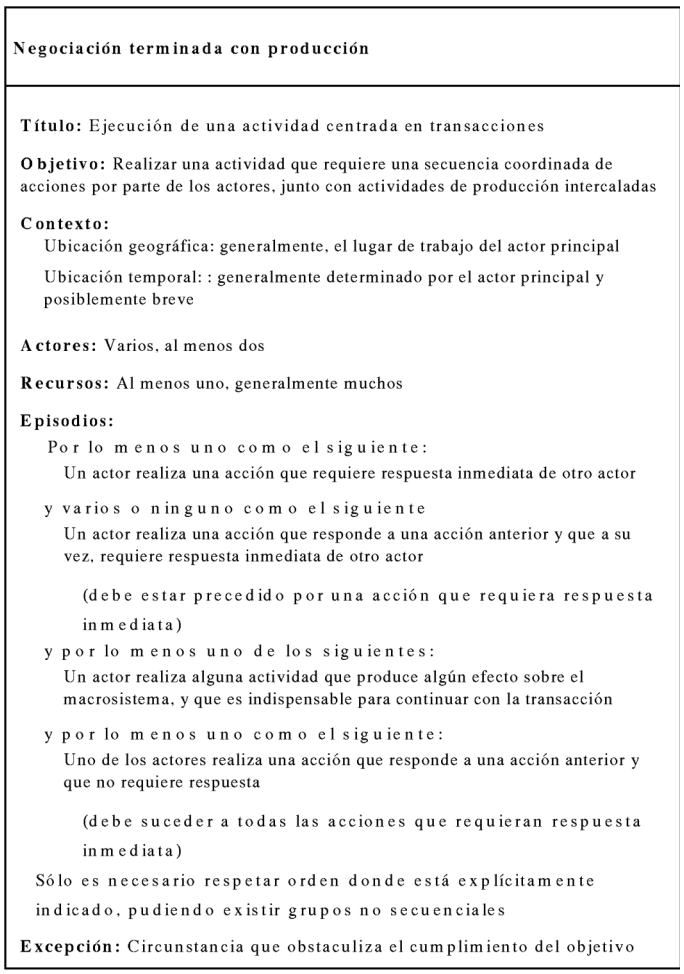
Além disto, observou-se que várias situações são compostas de diferentes tipos de episódios, ou seja, novos tipos de cenários:

- Produção + Serviço + Colaboração;
- Negociação inconclusiva com Produção ou Serviço ou Colaboração;
- Fim de negociação com Produção ou Serviço ou Colaboração;
- Etapa de Negociação com Produção ou Serviço ou Colaboração;
- Negociação terminada com Produção ou Serviço ou Colaboração;
- Negociação inconclusiva com disparo de cenários e Produção ou Serviço ou Colaboração;
- Etapa de negociação com disparo de cenários e Produção ou Serviço ou Colaboração.

Criada toda esta classificação e, com ela, os padrões de construção de cenários seguindo a estrutura definida por Leite [**?**] título, objetivo, contexto, atores, recursos, episódios e exceções. Além destes dados, foram acrescentados textos complementares. Por exemplo, quanto aos episódios, pode-se acrescentar uma descrição dos tipos de episódios, a quantidade de episódios de cada tipo, a ordem em que estão. Um exemplo pode ser visto na figura 3.3.1.

A partir dos métodos usuais de aquisição de dados para elicitação de requisitos (entrevistas, questionários, etc), definem-se situações compostas por vários episódios. Classificam-se estes de acordo com o número de atores envolvidos, se requerem ou não resposta (e, se for o caso, se a resposta é necessária imediatamente ou pode ser deferida) e, principalmente, pelo papel desempenhado pelos atores. Consequentemente, pode-se classificar as situações de acordo com

a classificação dos episódios que as compõem (ou seja, de acordo com um padrão). Apesar de existir um leque grande de padrões devido as combinações de tipos de episódios existentes em cada cenário, utilizando-se uma heurística baseada em árvores de decisão esta tarefe torna-se muito fácil.



Negociación terminada con producción

Título: Ejecución de una actividad centrada en transacciones

Objetivo: Realizar una actividad que requiere una secuencia coordinada de acciones por parte de los actores, junto con actividades de producción intercaladas

Contexto:
Ubicación geográfica: generalmente, el lugar de trabajo del actor principal
Ubicación temporal: : generalmente determinado por el actor principal y posiblemente breve

Actores: Varios, al menos dos

Recursos: Al menos uno, generalmente muchos

Episodios:
Por lo menos uno como el siguiente:
Un actor realiza una acción que requiere respuesta inmediata de otro actor
y varios o ninguno como el siguiente
Un actor realiza una acción que responde a una acción anterior y que a su vez, requiere respuesta inmediata de otro actor
(debe estar precedido por una acción que requiera respuesta inmediata)
y por lo menos uno de los siguientes:
Un actor realiza alguna actividad que produce algún efecto sobre el macrosistema, y que es indispensable para continuar con la transacción
y por lo menos uno como el siguiente:
Uno de los actores realiza una acción que responde a una acción anterior y que no requiere respuesta
(debe suceder a todas las acciones que requieran respuesta inmediata)
Sólo es necesario respetar orden donde está explícitamente indicado, pudiendo existir grupos no secuenciales

Excepción: Circunstancia que obstaculiza el cumplimiento del objetivo

Figura 4: Descrição do padrão Negociação Terminada com Produção.

No entanto, de nada adiantaria todo este esforço se não houvesse uma maneira viável de empregá-lo. Portanto, a seguinte heurística é empregada:

1. O primeiro passo é produzir uma primeira versão dos cenários a partir do léxico do domínio da aplicação. Propõe-se que este seja criado através observação, leitura de documentação, entrevistas, dentre outras técnicas possíveis. Enfim, definem-se várias situações que proverão um meio para verificação e validação dos cenários.
2. Identificam-se os atores do universo do domínio da aplicação e extraem-se os efeitos causados por estes atores. Cada um destes será um novo cenário que será incorporado a lista de cenários canditados.
3. Através de um sistema especialista (figura 3.3.1, tenta-se extrair o máximo possível de informação sobre os cenários candidatos.

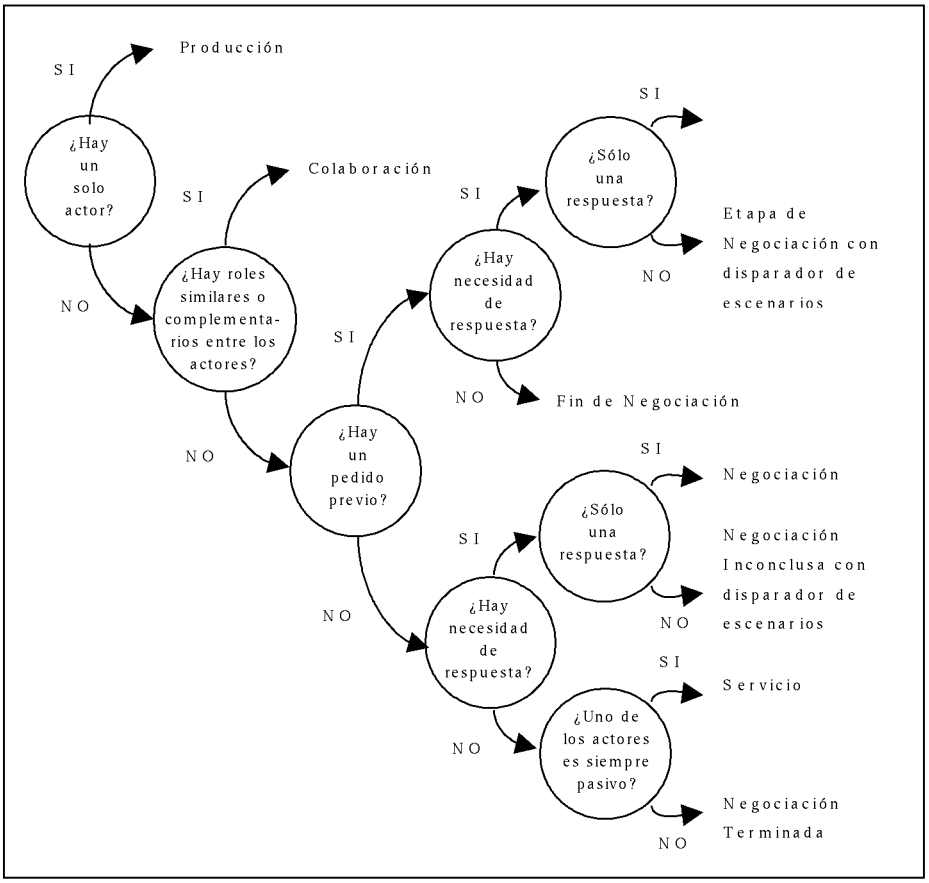


Figura 5: Árvore de decisão para a seleção de padrões.

A aplicação da heurística, conforme pode ser notado, não é complicada. Mas o mais importante é a validade da técnica. Esta possui características interessantes, tais como a possibilidade de reuso com um alto grau de abstração, logo no início do processo de engenharia de software. Isto permite que muitos erros sejam evitados ou detectados prematuramente, além de tornar o processo mais rápido. O fato de ser baseado em cenários restringe seu emprego a alguns poucos métodos de engenharia de software, felizmente ele pode ser utilizado neste em que se baseia a ferramenta deste projeto.

3.3.2 Identificação de Padrões de Reutilização de Requisitos de Sistemas de Informação

A aplicação de modelos e padrões de requisitos, uma vez padronizados quanto a maneira como são especificidados, permite identificar padrões de reutilização de requisitos, tanto os requisitos do cliente (requisitos-C) como dos requisitos do desenvolvedor (requisitos-D), permitindo assim um desenvolvimento mais rápido e eficiente do software. Outro fato é que, graças a rastreabilidade entre os requisitos-C, requisitos-D e elementos de mais baixo nível de abstração (tais como componentes de software), pode-se também reutilizar estruturas mais complexas tais como código fonte, ou seja, um reuso vertical, abrangendo diversos níveis de abstração do software.

Classifica-se os requisitos em requisitos-C (requisitos escritos/compreensíveis para o cli-

ente) e requisitos-D (requisitos feitos pelo desenvolvedor). Os requisitos-C podem ser de três tipos:

- Requisitos de informação: informações deve ser armazenada no sistema para satisfazer as necessidades dos clientes e usuários.
- Requisitos funcionais: casos de uso do sistema, contendo informações tais como o evento de ativação, as pré-condições, as pós-condições, os passos que compõe o caso de uso e suas exceções.
- Requisitos não funcionais: características não funcionais que o cliente e o usuário desejam no sistema.

Dentre estes tipo de requisitos-C, identificam-se vários padrões-R_c. No caso de requisitos de informação, por exemplo: cliente/sócio, produto/artigo, empregado, venda/fatura, fornecedor, pedido ao fornecedor, nota fiscal. Desta padrões, o que o ocorre com maior frequência é o primeiro, cliente-sócio (mais de 90% dos casos). Depois temos produto/artigo com 60% e assim por diante. Tão interessante quanto isto é o que estes padrões-R de requisitos de informação são diretamente utilizáveis, necessitando de mínimas modificações. O mesmo já não acontece com os padrões-R_c de requisitos funcionais. Estes são padrões baseados em parâmetros, o que demanda em um maior esforço para abstrair o padrão e, depois, os parâmetros que serão aplicados.

Além destes, temos os padrões de reutilização de requisitos-D (padrões-R_d). Eles sempre se relacionam com os seus respectivos padrões-R_c. A diferença entre um padrão e outro é o nível de detalhamento, trabalhando-se em um grau de abstração mais baixo. Os padrões-R_c para requisitos de informação são bem próximos de uma definição de classe, com a especificação explícita dos tipos de dados envolvidos. O mesmo acontece para os padrões-R_d para requisitos funcionais, utilizando OCL por exemplo.

3.3.3 Pontos de vista

A engenharia de requisitos orientada a ponto de vistas vêm do reconhecimento que os requisitos do sistema são gerados por várias fontes distintas e que tal realidade deve ser incluída explicitamente no processo. Esta visão não é absolutamente nova, na verdade desde o final da década de 70, com o SADT [**?**] e o SRD orr:1981, houve este reconhecimento. Porém, a utilização disto nunca ocorreu na proporção em que deveria, visto sua abrangência. Um exemplo isolado seria o CORE [**?**], utilizado pelo ministério de defesa da Inglaterra, do qual não se tem muitas informações nem ferramentas disponíveis a preços razoáveis.

Durante as últimas décadas, várias pesquisas foram desenvolvidas na área, surgindo vários modelos de ponto de vistas [**?**; **?**; **?**]. A origem de modelos diferentes surge das caracteristicas intrinsecas dos projetos para o qual o método foi criado, tomando definições de ponto de vistas que facilitassem o desenvolvimento dos sistemas. Por exemplo, em [SOMMERVILLE, 1996] são descritos os seguintes tipo de pontos de vistas:

- Uma fonte ou sumidouro de dados: Os pontos de vista são responsáveis por produzir ou consumir dados. Analisando o que é produzido e consumido, podemos detectar, por exemplo, dados gerados mas não utilizados e vice versa.

- Um framework para representação: Cada ponto de vista é considerado como um tipo particular de modelo do sistema (por exemplo, um modelo entidade-relacionamento, um modelo de máquina de estados, etc). Comparando-os, torna-se possível a descoberta de vários requisitos que não seriam detectados sem a utilização desta técnica.
- Um receptor de serviços: Os pontos de vista são externos ao sistema e recebem serviços deste.

Devido a natureza deste projeto, que visa uma ferramenta que suporte a definição dos requisitos, tendo como base um método de análise de requisitos voltado ao usuário (casos de uso), a concepção de um ponto de vista como um receptor de serviços é a mais apropriada.

3.3.4 REQAV: Modelo para Descrição, Qualificação, Análise e Validação de Requisitos

A necessidade de uma definição clara do software a ser construído é vital para o processo de engenharia de software. O REQAV é um modelo que aborda esse problema, propondo critérios de valor e peso à informação dos stakeholders para estabelecer condições de análise e validação dos requisitos.

O processo é composto por onze etapas, agrupadas em cinco fases: descrição do requisito, qualificação do requisito, qualificação da fonte de informação, aplicação de parâmetros de qualificação e composição do quadro de avaliação de risco de implementação do requisito.

A descrição dos requisitos consiste em planejamento, pesquisa inicial do material existente, identificação do stakeholder, descrição inicial dos requisitos, estruturação dos dados e composição da versão inicial do documento de requisitos. Ao final desta etapa, gera-se um documento preliminar de descrição de requisitos e um quadro descritivo de requisitos.

A fase de qualificação dos requisitos obtém a qualificação de cada requisito requisito e a relação de dependência entre eles, analisando, para isso, três aspectos: qualificação funcional, área de origem e a relação de dependência entre eles. Adotando uma qualificação variando de 1 a n, teríamos n^3 possíveis combinações (ou seja, n^3 níveis de qualificação do requisito).

A qualificação da fonte de informação obtém a qualificação do stackholder em função do seu ponto de vista, sua qualificação funcional na organização e a exigência da informação. O raciocínio segue o mesmo da qualificação dos requisitos (n^3 níveis de qualificação possíveis).

A aplicação de parâmetros de qualificação compreende a apropriação dos resultados das etapas de qualificação do requisito, qualificação da fonte de informação e o comparativo dos resultados para avaliação de risco.

Finalmente, temos a fase de composição do quadro de avaliação de risco. Esta consiste em, a partir da avaliação das informações obtidas na qualificação dos requisitos e das fontes de informações, juntamente com a aplicação de parâmetros de qualificação, gerar um quadro de avaliação de risco.

A aplicação do modelo proposto possui inúmeras vantagens:

- Os critérios adotados permitem uma visualização dos requisitos prioritários;
- Estes mesmos critérios possibilitam identificar requisitos que terão de ser revisados;
- A aplicação do modelo facilita a manutenção do foco durante o desenvolvimento do sistema;
- As informações geradas durante o processo servem de fundamento para a negociação dos requisitos.

3.3.5 Usando diferentes meios de comunicação na negociação de requisitos

Há tempos percebe-se a necessidade de aproximação dos clientes e desenvolvedores para definir os requisitos, principalmente para resolver os conflitos encontrados. Sempre imaginou-se que a maneira mais efetiva de fazê-lo era através de um encontro cara a cara entre as pessoas que vão negociar os conflitos. No artigo estudado [DAM2000], investiga-se a performance de grupo e relacionamento interpessoal na engenharia de requisitos distribuída, confrontando-se, então, a comunicação utilizando o computador e seus recursos multimídias (som e imagem) com a forma de comunicação que até então se acreditava ser mais efetiva.

Na pesquisa realizada, foram estabelecidos dois objetivos: o estudo do efeito da comunicação no desempenho do grupo na negociação de requisitos e os efeitos da configuração do grupo. Tomou-se como variáveis independentes o modo de comunicação e o arranjo do grupo, as variáveis dependentes foram o desempenho do grupo e percepção pessoal. Destas variáveis, a mais importante foi a de desempenho do grupo na negociação dos requisitos. Esta negociação pode ser distributiva (os conflitos são resolvidos através da eliminação de um, ou seja, o sistema atende somente uma parcela dos stakeholders) ou integrativa (os conflitos são negociados e, no fim, atende-se os requisitos de todos os envolvidos da melhor maneira possível).

O experimento consistiu na negociação de requisitos funcionais de um sistema de gerenciamento bancário. Estudou-se cinco configurações de grupo: uma cara a cara e outras quatro distribuídas. Abaixo temos as configurações utilizadas:

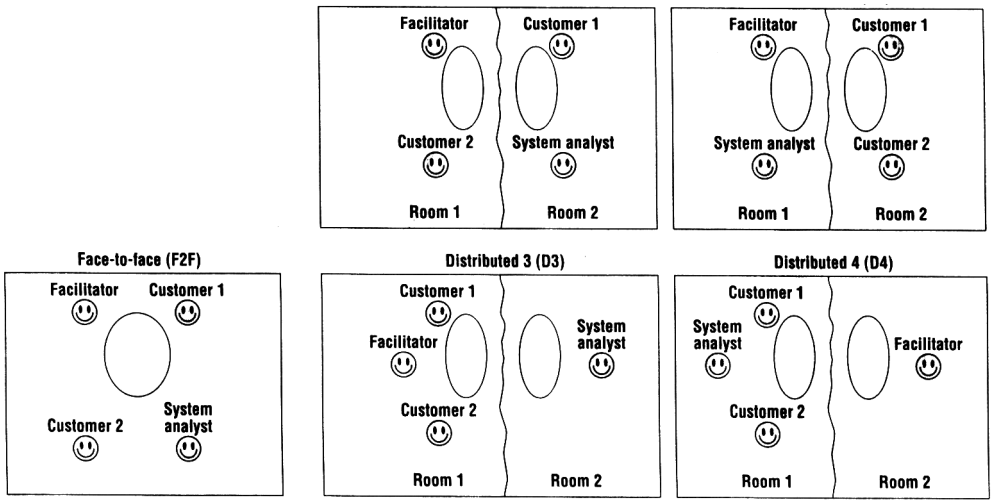


Figura 6: Cinco configurações de grupo: (a) cara a cara e (b) distribuída.

Os resultados do experimento são interessantes. Verificou-se que a comunicação utilizando o computador como meio é tão eficiente quanto e até melhor que a comunicação face a face. Mais ainda, observou-se em D1 os melhores resultados, melhores até que o F2F (que fora utilizado como referência para comparação). A explicação para este resultado é que, em D1, os stakeholders estavam separados. Outro fato interessante foi que em D2 e D3, no qual o analista de sistema está separado dos clientes e estes estão juntos, a negociação dos requisitos foi distributiva, conseqüencia da persuação que um cliente exerce sobre o outro, do proximididade entre as pessoas e relacionamentos interpessoais. Isto foi confirmado pela análise da percepção pessoal, na qual as pessoas, apesar de gostarem desta proximidade, mencionam que esta permite que uma pessoa influencie a outra mais facilmente, o que prejudica a realização da tarefa. A

reduzidade capacidade de perceber as emoções das pessoas, como no caso D1, permitiram aos clientes um melhor entendimento das necessidades, permitindo um raciocínio mais claro, além de possibilitar que o analista de sistema se mantenha mais imparcial.

3.4 Formatos para intercâmbio de modelos

Uma das maiores barreiras no desenvolvimento de software, principalmente agora com a participação de grupos geograficamente distantes, é o intercâmbio de dados entre as diferentes ferramentas utilizadas. Uma solução simples seria forçar a todos a utilização de uma mesma ferramenta. Claro que isto é impossível na maioria dos casos, além de ser pouco eficiente. O melhor seria a criação de um formato padrão para realizar esta troca de informações. Em [ST-DENIS, 2000], vários requisitos foram definidos quanto a tal padrão:

- Transparência: o processo de codificação/decodificação especificado pelo formato de intercâmbio de modelos não deve remover, adicionar ou alterar qualquer informação contida no modelo original.
- Independência de escala: deve ser adequado a projetos reais, de grande porte. Algumas características que devem ser observadas são a compressibilidade, a possibilidade de fazer o intercâmbio de modelos parciais (somente as diferenças entre dois modelos, por exemplo), criação de ligações (referências) entre os modelos (ao invés de duplicar os dados).
- Simplicidade: talvez o mais óbvio e, ao mesmo tempo, o mais difícil. O formato deve atacar a raiz do problema, o resolvendo de maneira eficiente (utilizando um mínimo de recurso computacional e humano). A simplicidade contribui também para uma menor complexidade das ferramentas necessárias para a manipulação dos dados gerados, além de reduzir a chance de ter erros no padrão.
- Neutralidade: garante que o padrão acomoda (ou simplesmente ignora) aspectos específicos da plataforma na qual está sendo utilizado. Por exemplo: linguagem, extensões da linguagem sendo transportada.
- Formalidade: A especificação deve estar definida formalmente, eliminando assim interpretações diferentes (e muitas vezes conflitantes) do padrão. Isto é vital para a construção de ferramentas que automatizem a aplicação do formato de intercâmbio de modelos.
- Flexibilidade: capacidade de acomodar os mais diferentes tipos de modelos, sejam estes completos ou incompletos.
- Capacidade de evolução: o formato deve ser capaz de atender futuros requisitos.
- Popularidade: se o formato não for aceito pela maioria, sua função principal deixa de existir.
- Completude: o formato deve ser completo o suficiente, evitando que os usuários (e ferramentas) tenham de incluir funções comumente usadas porém não diretamente suportadas pelo formato.
- Identidade com metamodelos: utilizar um metamodelo universal ou utilizar metamodelos específicos no processo.
- Reuso de padrões já existentes

- Legibilidade: apesar do formato ser destinado a manipulação por ferramentas, é desejável que ele seja legível o suficiente para que um engenheiro de software consiga entendê-lo e modificá-lo sem a utilização das mesmas.
- Integridade

De acordo com estes requisitos, pode-se identificar que vários podem ser atendidos com a utilização de XML (Extensible Markup Language): ela é simples, flexível, possui um bom suporte quanto a ferramentas, sua popularidade é extremamente elevada, possui boa legibilidade, reuso de padrões, neutralidade. A linguagem XML é bem recente, foi recomendada pela W3C em fevereiro de 1998, chegando a ser estranho uma linguagem tão recente ser tão popular. Mas, na verdade, os alicerces do XML são extremamente sólidos: ela se trata de um subconjunto da SGML (Standard Generalized Markup Language, ISO 8879:1986); todo o seu processo de criação foi acompanhado por empresas tais como IBM, Microsoft, todo o movimento opensource, sendo definida por uma organização sem fins lucrativos (W3C).

No entanto, somente a utilização da XML não é o suficiente. São necessários mecanismos para suportar diferentes metamodelos de maneira fácil. A primeira solução imaginada, a utilização direta de metamodelos em DTDs ou XML Schema, era pouco flexível. A OMG decidiu criar, então, o XMI.

Ele define regras para transformar metamodelos definidos em MOF em DTDs e, futuramente, XML Schemas. O MOF é a base na definição dos metamodelos utilizados pela OMG, logo todas as linguagens por ela definidas podem ser transportadas. Oficialmente, existem DTDs criadas para a UML e a MOF, mas nada impede que empresas criem suas próprias a partir de metamodelos definidos em MOF ou até mesmo UML, utilizando para isso os mecanismos do XMI.

O único problema atual desta tecnologia é que nem todas as empresas adotam o mesmo metamodelo oficial. Por exemplo, o Rational Rose 2000 utiliza um metamodelo da UML ligeiramente diferente daquele definido na especificação da UML 1.3. Outras ferramentas, tais como o ArgoUML, que aceitam arquivos neste formato (XMI), conseguem ler o arquivo corretamente, no entanto o modelo por eles apresentados não é idêntico àquele do Rational Rose. Em um teste realizado em agosto de 2000, utilizando as ferramentas ArgoUML 0.8, Rational Rose 2000 com suporte a XMI, MagicDraw UML 3.6 e Together 4.0. Observou-se que suportar ou não o XMI era inútil, porque os documentos produzidos eram, em sua grande maioria, incompatíveis entre as diversas ferramentas (foi possível ler um diagrama de caso de uso do Together 4.0 no Rational Rose, porém mesmo este caso de teste não foi 100% bem sucedido). Diante deste resultados, procurou-se informações mais aprofundadas sobre o assunto, utilizando principalmente listas de email. Nas duas listas consultadas[1], confirma-se esta situação (atualmente, no entanto, a compatibilidade parece ser maior do que na época em que o teste foi feito).

3.5 Persistência utilizando banco de dados

A necessidade por pesquisa dentre os casos de uso e pontos de vista do sistema tornam a utilização de persistência nativa do Java, baseada em serialização em arquivo, inadequada: a velocidade seria baixa, o acesso concorrente seria complexo. Após uma extensa pesquisa,

[1]Lista sobre XMI do Distributed System Technology Centre (xmi@dstc.edu.au) e Request Task Force do XMI (xmi-rtf@emerald.omg.org)

descobriu-se que o banco de dados PostgreSQL possuia mecanismos de serialiação compatíveis com o Java, porém armazenando os dados em sua base dados. Aliando a isto a possibilidade de criar um Corba Query Service utilizando este sistema de banco de dados, qualquer tipo de pesquisar a ser efetuada torna-se muito mais rápida e prática.

Primeiro, necessita-se de uma explicação sobre o PostgreSQL. Ele é um banco de dados gratuito, com código fonte disponível, licenciado segundo uma licença compatível com BSD. Trata-se de um banco objeto-relacional.

No entanto, somente estas características não possilitam a serialização. Necessita-se de um mecanismo que possibilita armazenar as referências a objetos que encontramos nos objetos Java. Felizmente o PostgreSQL permite isso, criando tabelas nas quais os campos podem ser nomes de outras tabelas. Por exemplo:

```
test=> create table users (username name,fullname text);
CREATE
test=> create table server (servername name,adminuser users);
CREATE
test=> insert into users values ('peter','Peter Mount');
INSERT 2610132 1
test=> insert into server values ('maidast',2610132::users);
INSERT 2610133 1
test=> select * from users;
username|fullname
--------+--------------
peter   |Peter Mount
(1 row)

test=> select * from server;
servername|adminuser
----------+---------
maidast   |  2610132
(1 row)
```

Na tabela "server", como pode ser notado, criou-se uma referência ao usuário peter da tabela "users"(que possui um número de identificação 2610132). Agora que é possível guardar, além de atributos como String, int, double, referências para outros objetos. Foi criado um novo componente para o microkernel, o DBPersistenceHandler, que extende a classe org.postgresql.util.Serialize, habilitando o microkernel a utilização de tal mecanismo de serialização.

3.6 Extensão da linguagem UML

As extensões da linguagem UML utilizadas na definição de requisitos são implementadas através do uso de esteriótipos e tagged values. Estes mecanismos de extensão da UML, definidos no pacote ExtensionMechanisms do Foundation Packages, foram feitos justamente para acomodar possíveis dados extras necessários aos modelos e processos de software. Outra possibilidade para acomodar as necessidades deste projeto seria criar uma extensão do metamodelo

da UML, definindo as novas metaclasses e metaconstrutores através da MOF. Porém, o uso dos mecanismos de extensão da UML são suficientes para atender as necessidades do projeto.

No pacote Extension Mechanisms da UML, temos definidos duas novas metaclasses: Stereotype, TaggedValue. Além disso, temos a metaclasse Constraint, do pacote Core. Estes mecanismos podem ser aplicados a qualquer ModelElement ou derivado deste, possuindo um valor semântico maior que qualquer outro mecanismo da UML (especialização, relacionamentos).

Um esteriótipo é uma metaclasse que altera a elemento do modelo de maneira que ele pareça ser uma instância de um metamodelo virtual (virtual porque ele não é definido da UML, mas parece que é). Pode haver, no máximo, um esteriótipo associado a um modelo de elemento. Todos os tagged values e restrições aplicados em um esteriótipo também são válidos no modelo de elemento, atuando assim como uma pseudo metaclasse descrevendo o o elemento. Outra característica interessante é que pode-se derivar um esteriótipo de outro esteriótipo (um esteriótipo é uma especialiação de modelElement).

Tagged values são propriedades arbitrárias associadas a um elemento do modelo, representadas por uma tupla (nome,valor). Pode-se associar qualquer número de tagged values a um elemento, salvaguardando a restrição destas serem únicas quanto a este.

Constraints permitem que sejam definidas restrições semânticas ao elemento do modelo, utilizando para isto uma linguagem, tal como a OCL (que foi feita especificadamente para isto), uma linguagem de programação, notação matemática, linguagem natural. Geralmente utilizam-se linguagens definidas formalmente, possibilitando assim uma aplicação deste regras através de ferramentas. Uma restrição pode possuir o atributo "body"e a associação "constrainedElement". O corpo "body"pode possuir uma expressão booleana que define a restrição. Esta expressão sempre deve ser verdadeira para instâncias de elementos com esta restrição quando o sistema está estável, ou seja, não está sendo feita nenhuma operação no sistema. Caso contrário, é dito que o modelo está mal formado. A associação "constrainedElement"é uma lista ordenada dos elementos do modelo sujeitos à uma restrição. Se o elemento em questão for um esteriótipo, todos os elementos que possuem aquele esteriótipo também estarão sujeitos a restrição.

Estudados os mecanismos de extensão da UML, definiu-se, então, as extensões da UML a serem utilizadas a fim de representarmos nossa linguagem. Escolheu-se pela utilização de tagged values. Para os atores, foram definidos os seguinte:

- isDistributed
- isParallel
- isExclusive

Para os casos de uso:

- isSequential
- isDistributed

E, finalmente, para os relacionamentos:

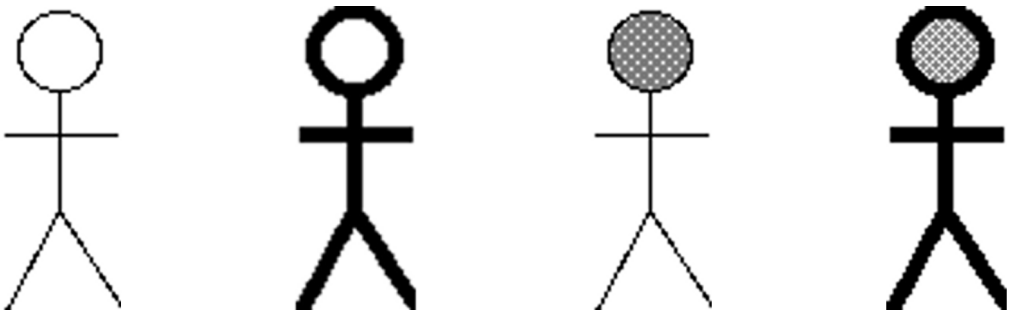
- isSequential
- isParallel

Uma questão observada durante a pesquisa realizada sobre UML foi a complexidade da linguagem. Esta questão já fora levantada inclusive pelo grupo que desenvolve a UML, a OMG, que, para sua próxima grande versão, a 2.0, pretende reduzir tal complexidade. Observou-se em listas de discussões sobre o assunto que, inclusive, diversas ferramentas estão utilizando modelos ligeiramente incompatíveis, o que acaba prejudicando o esforço de uma implementação completa da UML. Por isso, optou-se por guardar os dados inerentes aos níveis mais altos e somente aqueles mais utilizados.

(a) Relacionamento entre casos de uso

(b) Relacionamento entre casos de uso paralelos

Figura 7: Representação dos relacionamentos



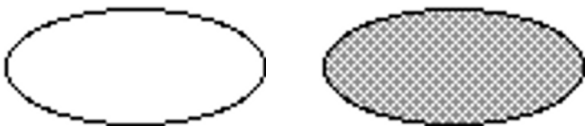
(a) Ator exclusivo

(b) Ator paralelo

(c) Ator distribuído

(d) Ator paralelo e distribuído

Figura 8: Representação dos atores



(a) Caso de uso sequencial

(b) Caso de uso distribuído

Figura 9: Representação dos casos de uso

4 Resultados e Discussão

4.1 Análise das diversas técnicas estudadas

Como pode ser notado, foi dada uma ênfase em abordagens que evitassem um alto grau de formalismo. Não que sua aplicação não seja válida, esta escolha se deve, principalmente, ao fato de sistemas distribuídos serem, geralmente, de grande porte, o que dificultaria sua definição de maneira exata e sem ambiguidade em todos os aspectos, o que torna difícil o emprego de tais técnicas.

Implementar uma ferramenta que automatize a definição de requisitos, de acordo com a metologia MDSODI, utilizando apenas uma das abordagens estudadas, talvez fosse insuficiente, para não dizer um desperdício. Por exemplo, a idéia de seleção de requisitos com base em conhecimento aplicada a pontos de vista permite a determinação de prioridades de maneira fácil e transparente, algo muito desejável em um sistema no qual teremos um universo de atores muitos distintos que geram uma quantidade impressionante de requisitos. Por mais que se queira, é impossível atender as necessidades de todos. Deve-se, portanto, se concentrar no foco do problema e, na medida do possível, atender as outras necessidades de menor prioridade.

A utilização de padrões de casos de uso permite casar modelos de projetos diferentes. Esta possibilidade de reuso tão prematura permitiria o reaproveitamento de arquiteturas, a descoberta de requisitos que tinham sido esquecidos. No caso de sistemas desenvolvidos com base em casos de uso (utilizando o Unified Process, por exemplo), o nível de reaproveitamento seria ótimo. No entanto, no artigo em que se propõe a utilização de tal técnica, a mesma é implementada através de um pequeno sistema especialista. Provavelmente, nos sistemas objetivados pela ferramenta proposta neste trabalho de graduação, os modelos de caso de uso com que iremos nos deparar serão muito grandes ou com um nível de complexidade tal que impediria a fácil utilização de uma solução como fora utilizada. O ideal seria a identificação automática deste, o que tornaria a complexidade da ferramenta um pouco além do esperado.

Temos na tabela abaixo uma breve comparação das diferentes técnicas analisadas. A técnica de ponto de vista se destaca não tendo nenhum ponto negativo. Todas as restantes possuem algumas características não satisfatórias, dificultando um pouco a escolha de uma técnica complementar. Considerando a utilização de pontos de vista e a consequente necessidade de filtrar os inúmeros pontos de vistas existentes (e visões geradas), a REQAV se destaca, podendo aplicar seus conceitos em um processo rápido e eficiente para o requerido processo.

4.2 Criação de um sistema gerenciador de conhecimento

O estudo sobre os diversos sistemas de gerenciamento de conhecimento, em especial o kMail, sugere a importância da adoção de sistemas que utilizem dados de uma base de conhecimento. Na ferramenta em questão, esbarram-se em alguns problemas. Utilizar a base de conhecimento para decidir que requisitos escolher quando houver conflitos ou até mesmo para adicionar novos requisitos de acordo com os já existentes seria extremamente complexo devido a subjetividade dos requisitos.

Uma melhor abordagem seria utilizar o sistema no processo de análise dos requisitos, gerando sugestões ou críticas durante a manipulação dos casos de uso, por exemplo. No entanto, esse modo de aplicação deve sofrer melhor pesquisa. A ferramenta utilizada neste projeto para modelar a ferramenta, o ArgoUML, faz uma análise em tempo de execução nestes moldes.

Técnica	Abrangência	Usabilidade	Implementabilidade	Disseminação
Ponto de Vista	Boa	Boa	Boa	Muito Boa
REQAV	Boa	Razoável	Ruim	Boa
Padrões na Construção de Cenários	Razoável	Boa	Ruim	Boa
Padrões de Reutilização de Requisitos	Boa	Ruim	Boa	Boa
Utilização de diferentes meios de comunicação	Boa	Boa	Ruim	Boa

Tabela 1: Comparação das técnicas estudadas.

Algumas críticas são interessantes, mas o problema é que a quantidade total é muito grande, ultrapassando a casa das centenas. É claro, a ferramenta não tem conhecimento dos perfis dos usuários, o que torna a técnica menos eficiente. Mas, num sistema interativo, mesmo empregando técnicas mais apuradas, seria um problema. Uma melhor solução seria a utilização de agentes que verificariam os produtos gerados, procurando por problemas e notificando os responsáveis. Estes agentes poderiam ser acionadas ou ao desejo do responsável (com o objetivo de ver os resultados da verificação imediatamente) ou então aleatoriamente, em períodos de inatividade do sistema.

Optou-se, por fim, em não empregar sistemas gerenciadores de conhecimento na ferramenta, porém projetando-a de maneira que o mesmo possa ser feito em trabalhos futuros.

4.3 Proposta de Processo de Engenharia de Requisitos

A ferramenta, sem uma processo de engenharia de software, não tem valor. Por isso, propõe-se o seguinte processo de engenharia de software, referente a engenharia dos requisitos:

1. A primeira etapa consistem em adquirir as visões dos stakeholders sobre o sistema. As visões são inseridas no sistema através de um formulário na Internet, feito em Java, que se comunica com o Servidor de Visões. Esta inserção pode ser feita também através da ferramenta que o engenheiro está utilizando. Observe que as visões podem conter vários anexos. Exemplos úteis seriam um documento de análise de requisitos de projetos anteriores, uma entrevista digitalizada. A identificação do tipo de dado anexado é feito através do seu tipo MIME, que deve ser informado no momento da criação da visão. Com base nesta informação, pode-se utilizar uma programa do sistema corrente para acessar este dado.
2. O próximo passo é a validação e qualificação das visões, passo este muito importante visto o volume de visões que um sistema pode conter. Utilizando a ferramenta, escolhe-se um critério e um mecanismo de qualificação. Esta é, na verdade, uma abstração. O mecanismo de validação pode ser uma consulta SQL, um motor de inferência, regras estáticas.

3. Escolhe-se, em seguida, as visões consideradas mais relevantes pelo sistema e faz-se a análise das mesmas, extraindo seus casos de uso, atores, requisitos não funcionais, criando diagramas de caso de uso.
4. Enfim temos uma parte crítica do processo, a resolução de conflitos. A ferramenta analisa os diversos casos de uso, atores e seus relacionamentos e lista os pontos em que há conflito. A solução destes envolverá a interação com o usuário. Graças as visões, podemos rastrear as visões que originaram os casos de uso e atores, podendo tirar dúvidas e, se necessário, interagir com os usuários, buscando o fim do conflito. Em métodos anteriores, este passo seria muito complicado, geralmente não há esta preocupação com rastreabilidade dos dados neste nível, visto o volume de dados usualmente obtidos.

O processo, em si, não segue esta linearidade. Existe uma dependência de cada etapa com sua anterior, mas uma vez cumprida todas as etapas, as mesmas podem ser repetidas em qualquer ordem.

4.4 Framework veryhot

A necessidade de criação de diagramas de casos de uso atendendo a notação utilizada neste trabalho e a possível futura necessidade de novos diagramas do mesmo tipo motivou a criação de uma pacote que facilite esta tarefa. Seu nome é veryhot. Na verdade, ele é uma evolução de um pacote Java desenvolvido em [**?**]. Fora feita uma reengenharia do mesmo, simplificando-o (diminuição de números de classes) sem perder suas características originais, facilitano futura manutenção.

O sistema possui quatro componentes principais:

- Figure: Figura que pode ser manipulada pelo pacote.
- DrawingPanel: Responsável por armazenar as figuras e desenhá-las adequadamente.
- Tool: Ferramentas que criam e manipulam as figuras contidas no DrawingPanel.
- ObserverArgument: Utilizado para repassar as alterações ocorridas em uma figura entre os diversos objetos que a observam.

Nos diagramas a seguir, pode-se observar a estrutura do pacote:

Tendo conhecimento do pacote, é possível enumerar alguma de suas características principais:

- Simplicidade: Para criar novas figuras (um Actor, por exemplo), basta extender o VectorFigure ou criar uma nova classe que especialize a Figure. Muito provavelmente só necessitará de alterações o método paint().
- Versatilidade das ferramentas: Elas podem modificar as figuras e não precisam se preocupar com o redesenho da figura, podendo o desenvolvedor concentrar-se na funcionalidade das ferramentas e não na apresentação da figuras que manipula.
- Sistema de notificação avançado: A comunicação sobre mudanças nas figuras é assíncrono, baseado em eventos, obtendo um desempenho bom ao mesmo tempo que é de fácil implementação. A utilização do ObserverArgument permite passar informações complexas entre os objetos.

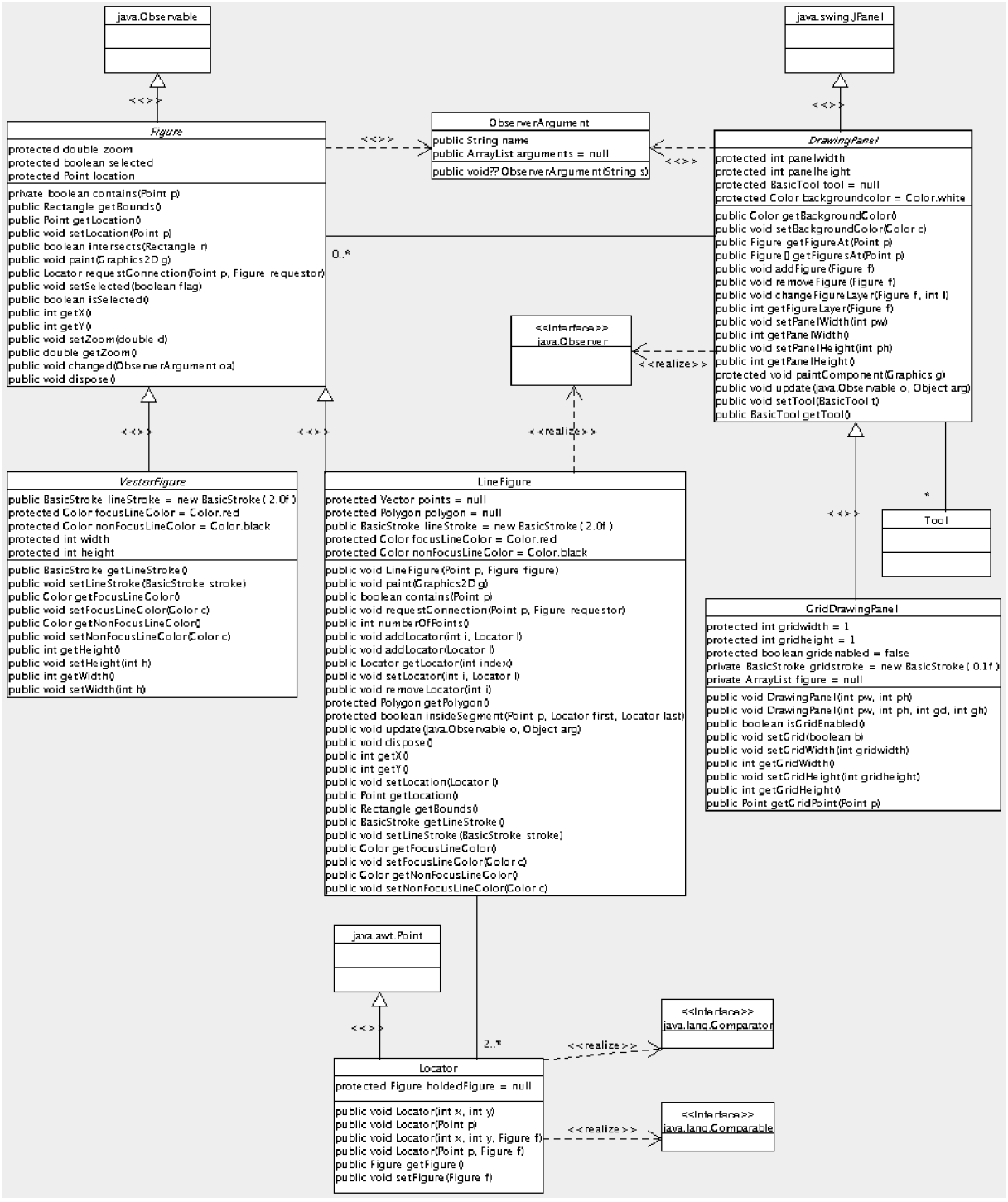


Figura 10: Diagrama de classes - Figures, DrawingPanel, ObserverArgument (package veryhot)

- Documentação: este framework está muito melhor documentado que a versão utilizada como base. Unindo a reengenharia com uma documentação mais detalhada, sua utilização ficou mais simples.

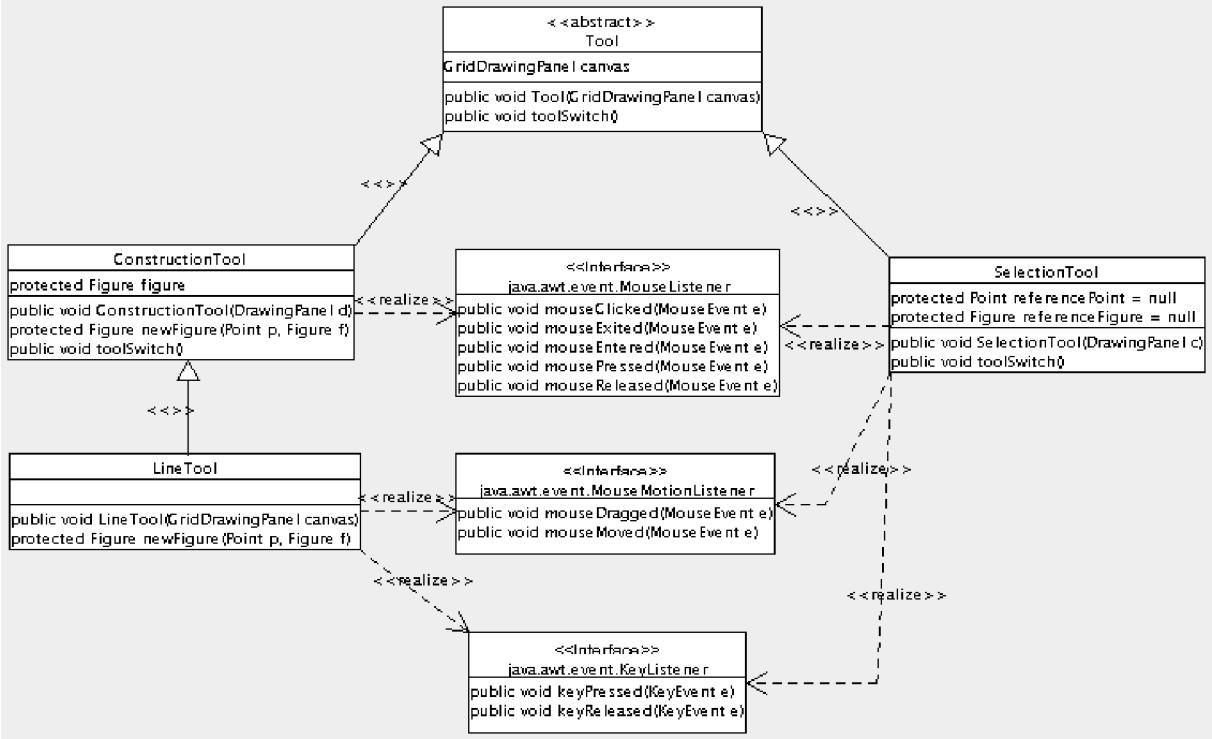


Figura 11: Diagrama de classes - Tool - (package veryhot.tool)

4.5 Microkernel

Novamente, reutilizando soluções criadas em trabalhos anteriores, decidiu-se por utilizar um microkernel para implementar os serviços básicos requeridos por cada serviço. As modificações feitas em relação ao anterior foi a remodelação do sistema quanto ao armazenamento de objetos, retirando componentes como Cache e PersistenceManager (responsáveis pelos serviços de cache e persistência), substituindo-os por um novo componente, MemoryManager, que armazena os objetos. Esta solução permite a utilização do sistema concorrentemente, ao contrário do sistema anterior. Uma possível futura melhoria seria a implementação de um sistema de cache que utiliza-se recursos avançados do Java para controle de objetos (utilizando-se das classes existentes em java.lang.ref).

O mecanismo de memória, MemoryManager, é uma interface para qual temos duas implementações: uma que armazena os objetos utilizando mecanismos de serialização para arquivo e outro que serializa para um banco de dados. A serialização para arquivo é a que temos no Java; a para banco de dados utiliza-se de recursos específicos do banco de dados PostgreSQL, já explicadas anteriormente.

Uma importante característica do MemoryManager é sua capacidade de selecionar objetos baseados em uma regra, característica esta necessária para a implementação da avaliação das visões, pontos de vistas e requisitos. Este mecanismo poderia, com as características atuais do microkernel, ser implementado a parte, como um outro serviço. No entanto, a performance de tal solução seria muito reduzida, optando-se portanto para este mecanismo.

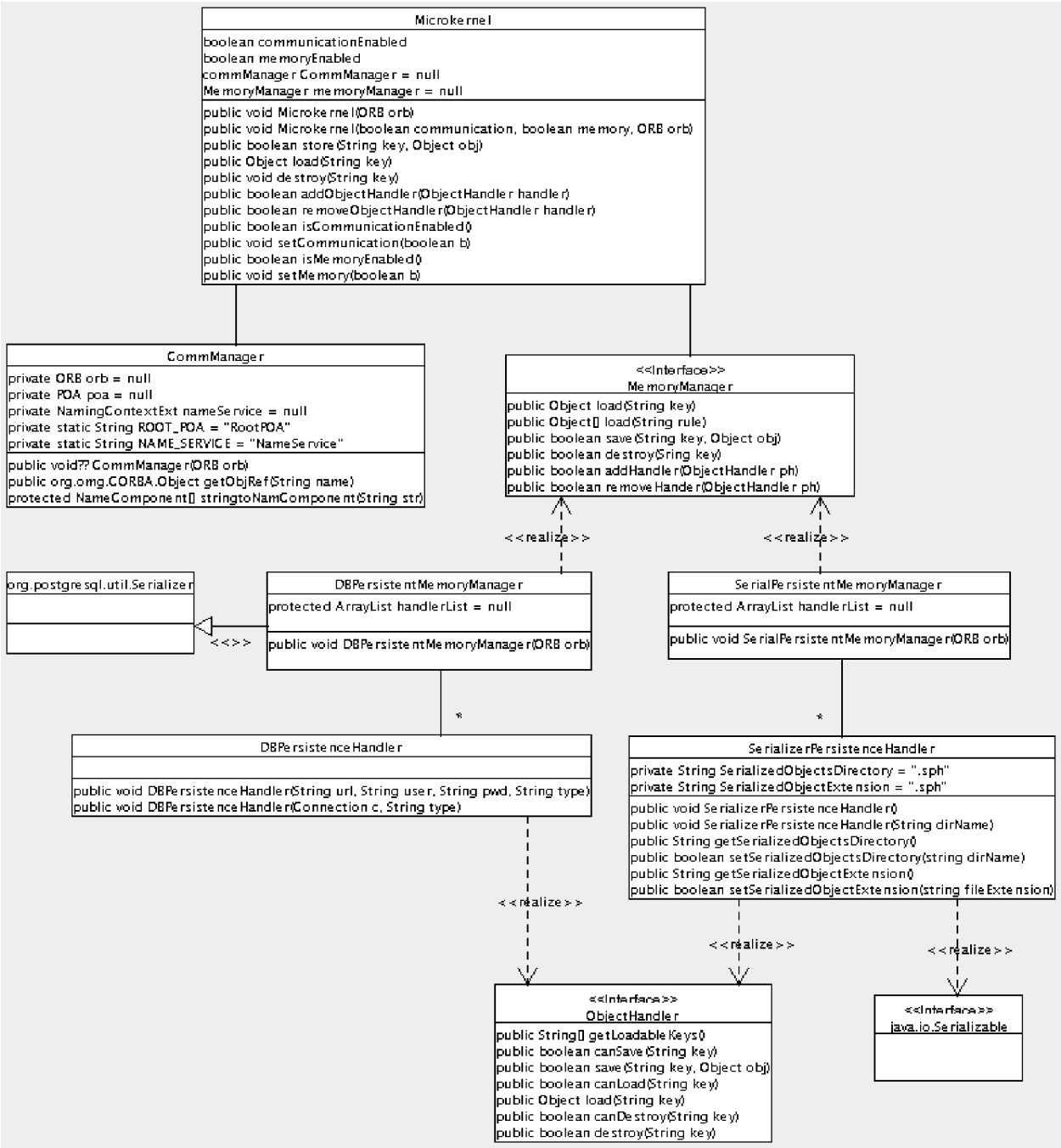


Figura 12: Diagrama de classes - Microkernel (package microkernel)

4.6 Arquitetura do Sistema

Em linhas gerais, a arquitetura do sistema pode ser descrita como mostrado na figura abaixo. A ferramenta, CoolCase, interage com todos os outros servidores:

- EntityServer: Contém todas as entidades referidas no sistema. Por exemplo, toda visão está relacionada com uma entidade.
- RequirementServer: Mantém um registro com os requisitos do sistema.

- UseCaseServer: Casos de uso.
- ActorServer: Atores.
- UseCaseModelServer: Armazena todos os relacionamentos entre casos de uso e atores em um sistema.
- UseCaseDiagramServer: Guarda diagramas de caso de uso.
- ViewServer: Gerencia as visões.

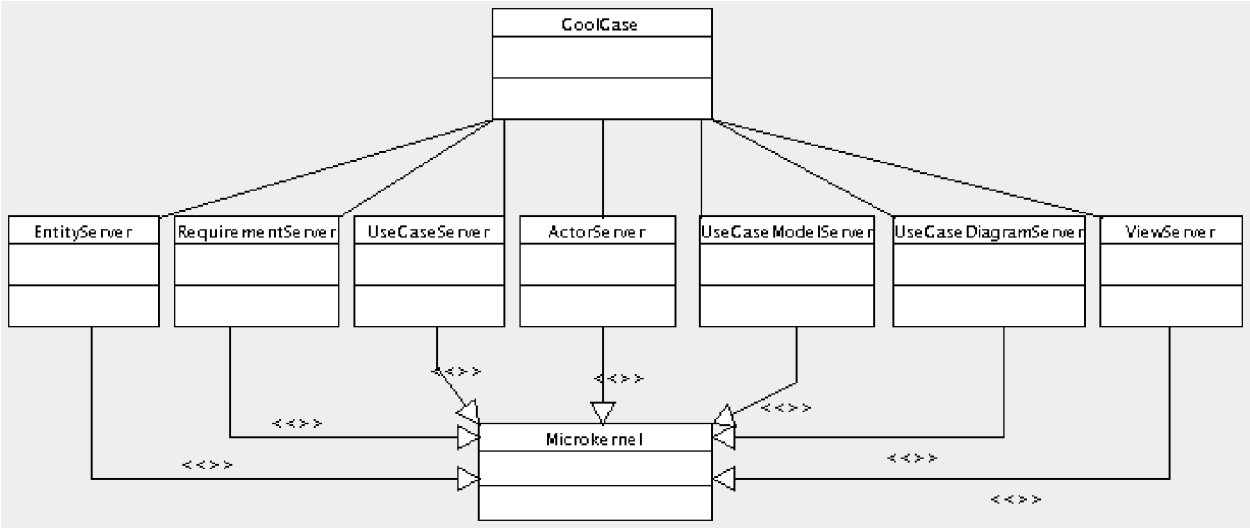


Figura 13: Arquitetura do sistema

4.7 Componentes do sistema

O sistema manipula sete tipos de componentes: entidades, requisitos, casos de uso, atores, modelos de caso de uso, diagramas de caso de uso e visões, todos eles armazenados e gerenciados por seus respectivos servidores. Em 4.7 pode-se observá-los. A fim de deixar o diagrama mais claro, foram omitidas o relacionamento que denota a especialização das classes em relação a classe Artefact.

4.7.1 Entidades

Toda visão está relacionada uma entidade, garantindo assim uma certa rastreabilidade. Mas o mais importante é a capacidade de avaliar as visões de acordo com as entidadas que as criaram. Para isso, armazenamos informações tais com papel desempenhado no ambiente do sistema sendo desenvolvido (role), habilidades, etc. Estes dados são armazenados em uma estrutura no formato (tipo do atributo, nome do atributo, valor). Toda entidade também possui um identificador único, no caso uma palavra.

No diagrama, pode ser notado que Entity é, na verdade, abstrata. A questão é que podemos ter entidades reais ou virtuais, tais como agentes inteligentes. Toda entidade deve possuir um mecanismo para ser notificada sobre alguma requisição (por exemplo, solicitar que a pessoa compareça a empresa para esclarecimentos sobre um determinado requisito).

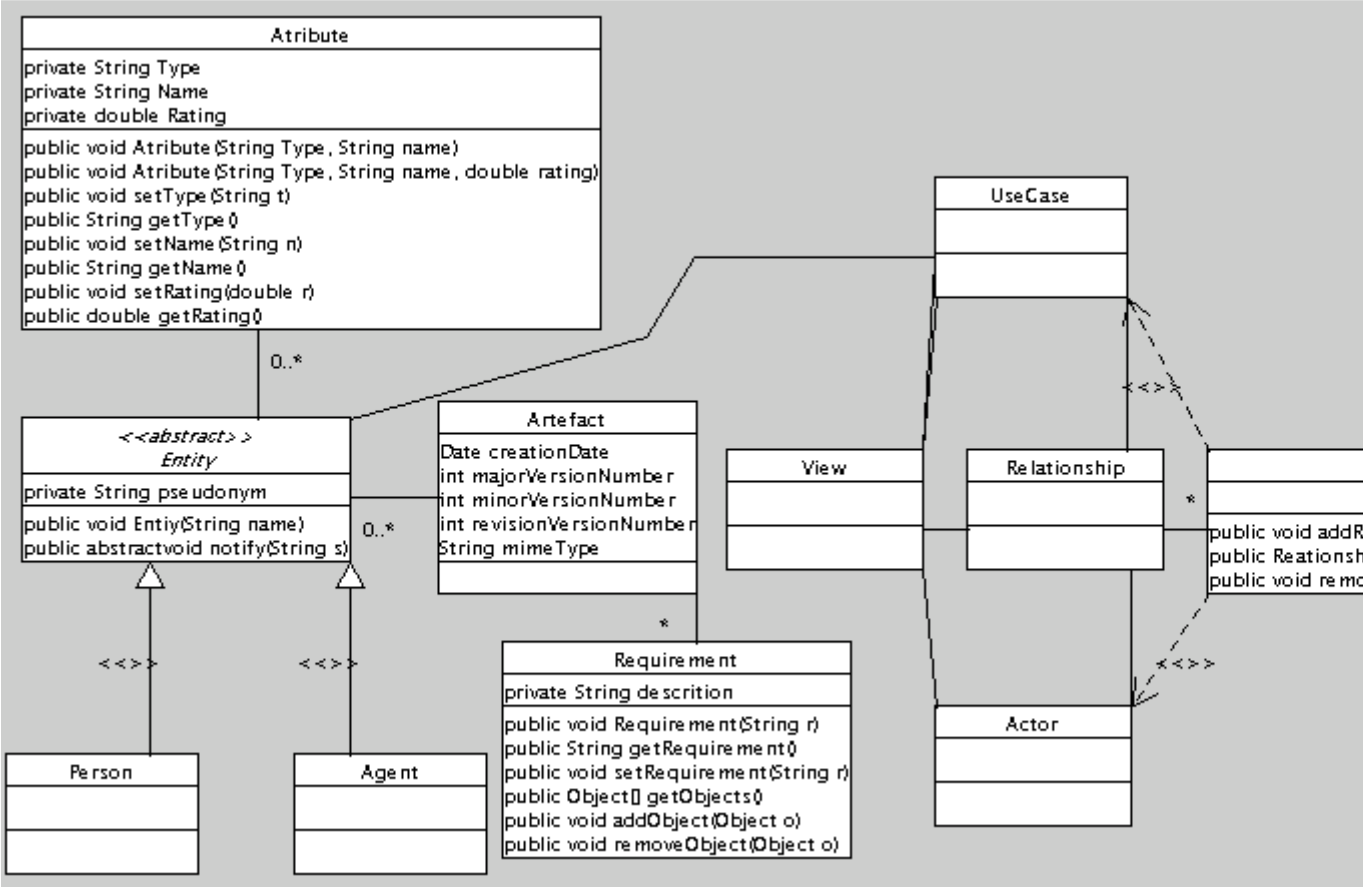


Figura 14: Componentes do sistema

4.7.2 Requisitos

Os requisitos são armazenados como um texto contendo referências para diversos atores, casos de uso e seus respectivos relacionamentos. De certa maneira, é o ponto no sistema em que o todo produzido durante o processo de engenharia de requisito é resumido, possibilitando uma rápida procura por algum requisito, avaliar o seu estado.

4.8 Casos de uso e atores

Estes seguem a estrutura definida na UML 1.3, contendo as restrições (em alguns aspectos) e melhorias (para suportar a descrição de características de sistemas distribuídos) conforme explicado em.

4.9 Modelo de caso de uso e diagramas de caso de uso

Os modelos de caso de uso são compostos pelos relaconamentos entre os casos de uso e atores. Na verdade, todo caso de uso e ator de um determinado projeto está incluído em um modelo de caso de uso. No entanto, visto a existência de servidores para estes, armazena-se somente seus relacionamentos, buscando os dados necessários nos servidores conforme requisitado.

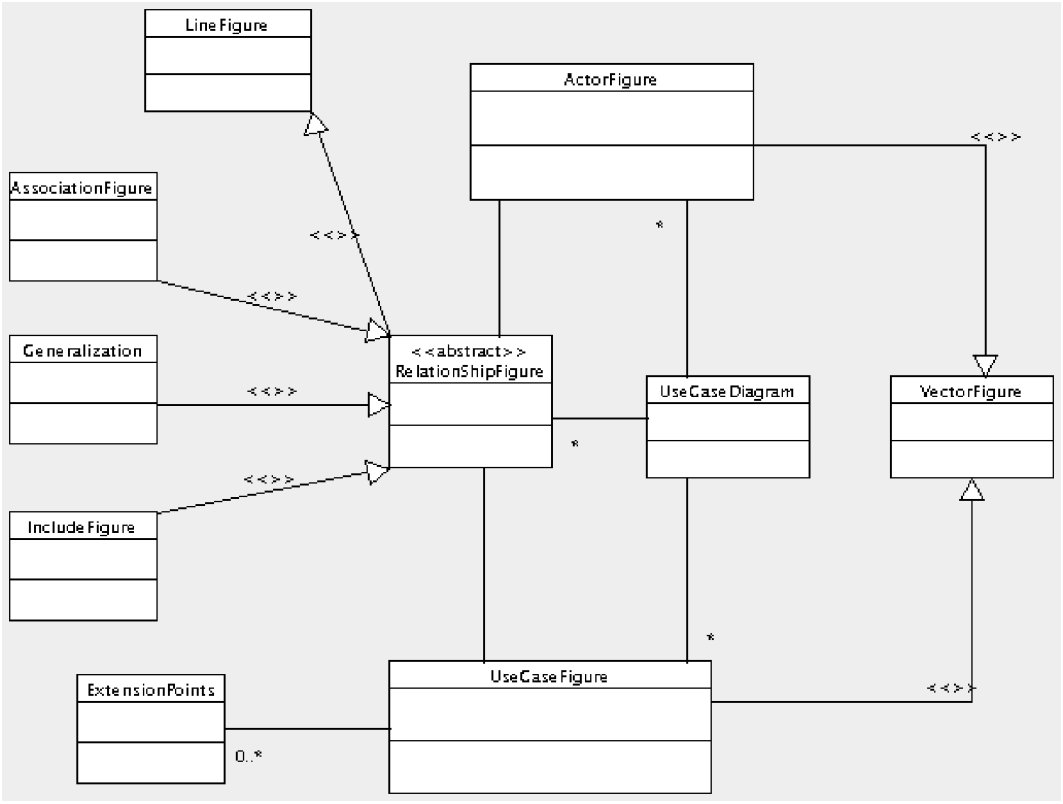


Figura 15: Diagrama de classe da CoolCase

Os diagramas de caso de uso retratam uma pequena parte do modelo de caso de uso. Alterações feitas em diagramas são refletidas no modelo e, portanto, deixá-lo inconsistente.

4.10 Visões

As visões são o meio de entrada dos dados necessários para a construção dos requisitos. Seus atributos são:

- Assunto
- Grau de importância (do ponto de vista da entidade que a enviou);
- Anexos
- Foco (a que componente ou aspecto específico esta visão está tratando)

5 CoolCase

Denominou-se CoolCase a parte da ferramenta com a qual o engenheiro irá trabalhar. Neste serão criados e editados os diagramas, sendo possivelacessar, através dela, as visões, requisitos, enfim, todos os elementos citados neste projeto. Em 5 pode-se observar o diagrama de classes da ferramenta. Como todo o resto do sistema, fora implementada em Java.

Em 5, pode-se visualizar um protótipo da ferramenta em execução.

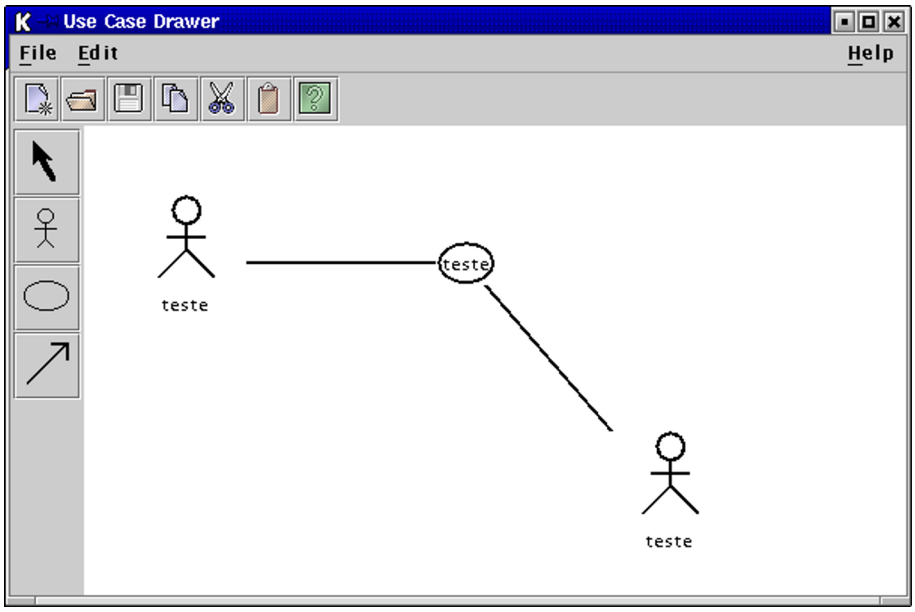


Figura 16: Protótipo da ferramenta em funcionamento

6 Conclusão

A área de engenharia de requisitos ainda está em suas fases iniciais de desenvolvimento, ao menos quando comparada a outras áreas da engenharia de software. Sua proximidade com o ser humano, conflitando com a exatidão requerida pela máquina, torna o processo muito complexo, surgindo, portanto, a necessidade do desenvolvimento de técnicas e ferramentas para torná-lo mais eficiente. Várias técnicas foram estudadas neste projeto, algumas fugindo do escopo usual de engenharia (por exemplo, a técnica que aborda os meios de comunicação para com o usuário) mas que acabam desempenhando um papel importante no processo. Em geral, observou-se que técnicas que utilizam-se mais dos dados sobre os stakeholders, valorizando o aspecto "informal"(formal o suficiente para a máquina e informal - natural - para o ser humano) possuem uma aceitação crescente na última década, muito provavelmente por causa da mudança do enfoque do software, cada vez mais destinada a um usuário comum, que não possui conhecimento para descrever formalmente um problema, desejando apenas a solução para o seu problema. A necessidade de lançar produtos ao mercado rapidamente também favorece esta técnicas, cuja facilidade de aplicação é muito superior a métodos de especificação formais de requisitos.

As técnicas abordadas, em especial a utilização de pontos de vista e a avaliação de requisitos, conseguem melhorar o processo de engenharia de requisitos. A associação do conhecimento sobre as entidades permite que o engenheiro entenda melhor as intenções do stakeholder e, na pior das hipóteses, vai garantir uma interação mais tranquila e rápida entre estes caso seja necessário consultá-los para resolução dos conflitos. A avaliação da importância das visões permitem filtrar os dados, evitando que o tempo caro dos engenheiros sejam dispensados em partes de pouca significância para o software sendo desenvolvido. O problema desta técnica são os filtros que serão utilizados: seus critérios devem ser justos, tentar atender o mínimo de visões que representem o máximo do domínio da aplicação. Os mecanismos apresentados no trabalho, através de consultas simples, obviamente são insatisfatórios. Modelos que utilizem mecanismos de inferência avançados e regras complexas e abrangentes devem ser desenvol-

vidos. Um grande problema neste aspecto é que o volume de dados gerados no processo de engenharia de requisitos é muito grande, causando que muitos recursos computacionais sejam dispendidos, tornando inviável sua utilização. O desenvolvimento de um sistema, baseado em banco de dados, com a flexibilidade dos sistemas inteligentes atuais, é necessária para o sucesso da aplicação da técnica em grande escala.

A complexidade da ferramenta superou a expectativa, atrasando o desenvolvimento da ferramenta que não se encontra com todas suas funcionalidades implementadas. Consequentemente, não foi possível realizar os estudos de caso planejados. No entanto, toda o projeto da ferramenta encontra-se pronto, grande parte deste já codificado, o que torna possível a continuidade da ferramenta para futuras pesquisas.

7 Bibliografia

Referências

BORTOLI, L. A.; PRICE, A. M. Um método para auxiliar a definição de requisitos. In: . Cancun-México: IDEAS 2000: Jornada Ibero Americana de Ingeneria de Requisitos Y Ambientes de Software, 2000. p. 1–12.

BOSAK, J. et al. **Extensible Markup Language (XML) 1.0**. First ed. [S.l.], February 1998.

CALDWELL, N. H. M. et al. Web-based knowledge managment for distributed design. **IEEE Intelligent Systems**, p. 40–47, May/June 2000.

DíAS, I.; METTEO, A. Objectory process stereotypes. **JOOP**, p. 29–38, June 1999.

DIAZ, J. S.; FERRAGUD, V. P.; PELOZO, E. I. Un entorno de generation de protótipo de interfaces de usuário a partir de diagramas de interacción. In: . Cancun-México: IDEAS 2000: Jornada Ibero Americana de Ingeneria de Requisitos Y Ambientes de Software, 2000. p. 145–154.

FOWLER, M.; SCOTT, K. **UML Distilled: Applying the Standard Object Modeling Language**. [S.l.]: Addison Wesley, 1997.

FOWLER, M. **UML distilled: a brief guide to the standard object modeling language**. Second ed. [S.l.]: Addison Wesley Longman, 2000. (Object Technology).

GRAVENA, J. P. Aspectos importatnes de uma metodologia para desenvolvimento de software com objetos distribuídos. 2000.

HAHN, J.; KIM, J. Why are some diagrams easier to work with? effects of diagrammatic representation on the cognitive integration process of system analysis and design. **ACM Transations on Computer-Human Interaction**, v. 6, n. 3, p. 181–213, September 1999.

HUZITA, E. H. M. Uma metodologia para auxiliar o desenvolvimento de aplicações para processamento paralelo. 1995.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. Second ed. Massachusetts: Addison Wesley Longman, 1999. (Object Tecnology).

KNAPIK, M.; JOHNSON, J. **Developing Intelligent Agents for Distributed System: Exploring Architecture, Technologies and Applications**. EUA: McGraw Hill, 1998.

LEITE, J. Enhacing a requirements baseline with scenarios. **Requirements Engineering Journal**, v. 2, n. 4, 1997.

O'LEARY, D. E. Using ai in knowledge managment: Knowldege bases and ontologies. **IEEE Intelligent Systems**, v. 13, n. 3, p. 34–39, may/june 1998.

PFAFFENSELLER, M.; PFAFFENSELLER, M.; KROTH, E. Uma ferramenta de apoio ao gerenciamento de componentes. 2001.

RABARIJAONA, A. et al. Building and searching an xml-based corporate memory. **IEEE Intelligent Systems**, p. 56–63, May/June 2000.

RIDAO, M.; DOORN, J.; PRADO LEITE, J. C. S. do. Uso de patrones en la construcción de escenarios. In: . [S.l.]: WER, 2000. p. 140–157.

SATO et al. Onix: An environment for the development of parallel object oriented software. In: **Internacional Workshop on High Performance Computing**. São Paulo: [s.n.], 1994. p. 167–183.

SCHELEBBE, H. **Distributed PROSOFT**. Germany, 1995.

SCHWARTZ, D. G.; TE'ENI, D. Tying knowledge to action with kmail. **IEEE Intelligent Systems**, p. 33–39, May/June 2000.

SOMMERVILLE, I. **Software Engineering**. Fifth ed. Massachussets: Addison Wesley, 1996. (International Computer Science).

SOMMERVILLE, I.; SAWYER, P. Viewpoints: principles, problems and a pratical approach to requirements engineering. 1997.

SOUZA, C. T.; OLIVEIRA, M. Ábaco. Um ambiente de desenvolvimento baseado em objetos distribuídos configuráveis. In: **XII Simpósio Brasileira de Engenharia de Software**. Maringá-PR: [s.n.], 1998. p. 205–220.

ST-DENIS, G.; SCHAUER, R.; KELLER, R. K. Selecting a model interchange format. In: IEEE (Ed.). [S.l.: s.n.], 2000.

SZYKMAN, S. et al. Design repositories: Engineering design's new knowledge base. **IEEE Intelligent Systems**, p. 48–55, May/June 2000.

TORO, A. D. et al. Identificación de patrones de reutilización de requisitos de sistemas de información. In: . [S.l.]: WER, 2000.

WOOLDRIDGE, M. J.; JENNINGS, N. R. Software engineering with agents: Pitfalls and pratifalls. **IEEE Internet Computing**, v. 3, n. 3, p. 20–27, may-jun 1999.

ZANLORENCI, E. P.; BURNETT, R. C. Reqav: Modelo para descrição, qualificação, análise e validação de requisitos. In: **IDEAS2000: Jornada Ibero Americana de Ingeneria de Requisitos Y Ambientes de Software**. [S.l.: s.n.], 2000. p. 61–72.

8 Anexos