LUCIO GERONIMO VALENTIN

UMA ARQUITETURA DE SOFTWARE PARA SISTEMAS DE DESCOBERTA DE CONHECIMENTO EM BANCO DE DADOS

MARINGÁ

2006

LUCIO GERONIMO VALENTIN

UMA ARQUITETURA DE SOFTWARE PARA SISTEMAS DE DESCOBERTA DE CONHECIMENTO EM BANCO DE DADOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Maria Madalena Dias

MARINGÁ

2006

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR., Brasil)

V156u

Valentin, Lucio Geronimo
Uma arquitetura de software para sistemas de descoberta de conhecimento em banco de dados / Lucio Geronimo Valentin. -- Maringá : [s.n.], 2006.
153 f. : il. color., figs., tabs.

Orientador : Prof.ª Dr.ª Maria Madalena Dias.
Dissertação (mestrado) - Universidade Estadual de Maringá. Programa de Pós-Graduação em Ciência da Computação, 2006.

1.Arquitetura de referência. 2. Arquitetura de software. 3. Sistema KDD. 4. Data warehouse. 5. Data mining. I. Universidade Estadual de Maringá. Programa de Pós-Graduação em Ciência da Computação.

CDD 21.ed. 005.74
006.4

LUCIO GERONIMO VALENTIN

UMA ARQUITETURA DE SOFTWARE PARA SISTEMAS DE DESCOBERTA DE CONHECIMENTO EM BANCO DE DADOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em 30/10/2006.

BANCA EXAMINADORA

Profª. Drª. Maria Madalena Dias
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Donizete Carlos Bruzarosco
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Aran Bey Tcholakian Morales
Universidade Federal de Santa Catarina – EGC/UFSC

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter criado este mundo e nele me colocado para que eu pudesse gozar do meu livre arbítrio. E além disto, por ter me dado mais uma chance de aqui permanecer por mais algum tempo que só Ele sabe.

À minha mãe que não mediu esforços para que um dia eu pudesse cursar uma universidade e neste momento estar concluindo o curso de mestrado.

À minha orientadora pelo seu apoio e sua presença em um dos momentos mais difíceis de minha vida.

À secretária Inês por ter ido além do seu secretariado e se mostrado como uma amiga, me auxiliando nos momentos que estava com dificuldades de locomoção.

A toda a coordenadoria do Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, pela compreensão, apoio e oportunidade de estar concluindo este curso.

Aos meus amigos da Incubadora que diretamente contribuíram com este trabalho. São eles: Antonio Alves, Márcia Tanimoto, Marcos Silvano, Tatiana Yuka, Juliana e André Moraes.

Aos meus amigos Guilherme e Maura pelo incentivo e pelas orações para que eu pudesse concluir mais esta fase de minha vida.

Ao meu amigo Ricardo pelo empréstimo do computador para término da redação deste trabalho.

Ao Conselho Nacional de Desenvolvimento Cientifico e Tecnológico - CNPq, pelo ininterrupto apoio financeiro.

A todos que direta ou indiretamente contribuíram para a realização deste trabalho.

Resumo

Atualmente, as rotinas operacionais das empresas em geral já se encontram quase que totalmente automatizadas. No entanto, geralmente, os sistemas computacionais existentes não estão integrados, gerando informações duplicadas e inconsistentes. Esta situação dificulta a busca por informações necessárias e confiáveis para a tomada de decisão. As tecnologias de *Data Warehouse* e Mineração de Dados (*Data Mining*) surgiram para solucionar este tipo de problema. *Data Warehouse* possibilita a integração e a organização dos dados existentes e a mineração de dados oferece técnicas para busca de conhecimentos interessantes. Existem trabalhos propostos na literatura que, geralmente, não consideram todo o processo de descoberta de conhecimento, ou ainda, apresentam o modelo sem se aprofundar nas especificações de implementação. Alguns deles estão direcionados à construção de um *Data Warehouse* e outros à aplicação de técnicas de mineração de dados. Existem, também, ferramentas OLAP que não abrangem as atividades de preparação de dados. Este trabalho apresenta uma arquitetura de referência, uma arquitetura de software e um *framework* que definem os componentes necessários para a implementação de sistemas de descoberta de conhecimento em banco de dados. São descritos os passos desde a construção do modelo de referência até os detalhes de implementação do *framework*. Desde a definição até a implementação são utilizados alguns padrões de projetos e melhores práticas de desenvolvimento.

Palavras chaves: Arquitetura de Referência, Arquitetura de Software, Sistema KDD, Data Warehouse e Data Mining.

Abstract

Currently, the operational routines of the company already are almost totally automatized. However, the existing computational systems are not integrated, generating duplicated and inconsistent information. This situation makes it difficult the search for necessary and trustworthy information for the taking of decisions. The technologies of Data Warehouse and Data Mining had appeared to solve this type of problem. Data Warehouse makes possible the integration and organization of the existing data. Data Mining offers techniques for search of interesting knowledge. Some works in literature, generally, do not consider the whole process of knowledge discovery or still, they present the model without if going deep the implementation specifications. Some of them are directed to the construction of one Data Warehouse and others to the application of techniques of data mining. There are OLAP tools that do not enclose the activities of preparation of data. This work presents a reference architecture, a software architecture and a framework that define the components necessary for the implementation of knowledge discovery systems in data base. The steps since the construction of reference model until the implementation details of framework are shown. Since the definition until the implementation some standards of best practices in projects and in development are used.

Key words: Reference Architecture, Software Architecture, KDD system, Data Warehouse and Data Mining.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ADC - Ambientes de Descoberta de Conhecimento

BTE - Busca, Transformação e Estágio

CRUD – *Create, Retrieve, Update and Delete*

DASD - *Direct Access Storage Device*

DW - *Data Warehouse*

ET – Extração e Transformação

ETL – *Extraction, Transformation and Load*

J2EE – *Java 2 Platform, Enterprise Edition*

L4G - Linguagem de quarta geração

KDD – *Knowledge Discovery in Database*

MD – Mineração de Dados

MIS - *Management Information System*

MVC – *Model-View-Controller*

OLAP - *On-line analytical processing*

OLTP – *On-line transactional processing*

PC - *Personal Computer*

SAD - Sistema de Apoio à Decisão

SBC - Sistemas Baseados em Conhecimento

SGBD – Sistema Gerenciador de Banco de Dados

UML – *Unified Modeling Language*

1. – Introdução

Com a evolução da computação e o seu crescente uso nas diversas áreas do conhecimento, os sistemas informatizados estão se tornando cada vez mais complexos. O desenvolvimento de hardware mais rápido e com menor custo foi alterando, com o decorrer do tempo, a maneira pela qual um software é desenvolvido. Novas metodologias e ferramentas são apresentadas de tempo em tempo buscando uma maneira rápida e eficiente para produção de software com qualidade e com custo reduzido. Com este desenvolvimento tecnológico, houve também uma alteração na organização dos mercados mundiais, a informação passou a ser um bem valioso para uma nação. De acordo com Pinheiro (2003), dentro deste cenário, a busca por conhecimento tornou-se uma necessidade cada vez mais presente nas instituições governamentais e no setor privado.

Dentro dessas instituições começaram a ser desenvolvidos sistemas cada vez maiores e mais integrados, resultando na manipulação de um enorme volume de informações. Em muitas empresas essas informações são utilizadas somente nos seus setores de produção e gerencial. Porém, se faz cada vez mais necessária, num mercado disputado e globalizado, a obtenção de conhecimentos que podem ser gerados com a análise sistemática de seus atuais bancos de dados. Instituições de pesquisa estão buscando informações que melhor direcionem os investimentos e apóiem na tomada de decisão.

Várias pesquisas estão sendo desenvolvidas nesta área de busca de informação em grandes volumes de dados. Dentre elas tem-se: *Data Warehouse* (DW) e Mineração de Dados (MD). Para DW não é encontrada uma tradução utilizada para a língua portuguesa. Ela envolve atividades de busca de grande volume de dados em ambientes operacionais heterogêneos, coletando e organizando os dados em um ambiente mais homogêneo e de acesso mais fácil para realização de consultas. MD tem as suas atividades representadas pelo seu próprio nome, trata-se de atividades de mineração de dados, ou seja, busca de informações que não são tão evidentes em determinado conjunto de dados pré-processados. O número de eventos e artigos produzidos nestas duas áreas tem crescido anualmente, mostrando como estas tecnologias estão sendo bastante aceitas e utilizadas (Cherobino, 2006).

Como evolução das ferramentas para atividades específicas do processo de busca de informações, as novas propostas estão sendo direcionadas à criação de Sistemas de

Descoberta de Conhecimento (sistemas KDD – *Knowledge Discovery in Database*). Esses sistemas buscam disponibilizar e integrar todos os recursos necessários para as atividades de transformação de dados brutos em conhecimento útil. Funcionalidades para localização, carga, preparação dos dados e ferramentas para execução de análises são disponibilizadas por esses sistemas. Um outro objetivo desses sistemas é, através desta integração, facilitar a descoberta de conhecimento e a utilização desta importante ferramenta na administração de empresas e instituições (Dias, 2001).

Devido ao alto custo e ao tempo de construção de um DW, somente grandes instituições têm usufruído comercialmente destes ambientes. Segundo Vassiliadis (2000), o projeto de um DW, por exemplo, é considerado de alto risco, por ter uma média de tempo para sua construção de 12 a 36 meses e a média de custos para sua implementação estar entre 1 a 1,5 milhões de dólares. E ainda, a construção de um DW exige o envolvimento de especialistas do negócio, especialistas em banco de dados, além de projetistas de sistemas. Em muitos casos, um DW é desenvolvido dentro da instituição utilizando diferentes ferramentas que, muitas vezes, não se integram e resolvem pequenas partes do problema. Os processos de localização, carga e transformação dos dados são realizados de forma bastante manual e com baixo índice de reutilização dos componentes desenvolvidos. Desta forma, o uso do DW fica bastante limitado e sua manutenção pode se tornar inviável.

A integração de um DW dentro de um sistema KDD permite que todas as informações sobre processo de composição do DW sejam armazenadas e reutilizadas. Vários trabalhos têm sido propostos e alguns modelos de arquitetura têm surgido, como por exemplo, o I-MIN (Gupta et al., 2004), Oracle Warehouse Builder (Oracle, 2005), AMORE (Psaila, 1998 apud Gupta et al., 2004) e Discovery board (Reinartz, 1999 apud Gupta et al,, 2004).

1.1. Objetivos

Este trabalho tem como objetivo geral definir uma arquitetura para sistemas de descoberta de conhecimento em banco de dados que integre todas as funcionalidades em uma única ferramenta e que possa ser utilizada na execução de diversos processos KDD definidos na literatura. Além disso, busca-se preencher algumas lacunas dos atuais trabalhos, principalmente no que se refere a uma definição detalhada do modelo de referência da aplicação separada da arquitetura de software a ser implementada, que às vezes são apresentadas sem uma separação clara.

As características arquiteturais de um modelo são extremamente importantes para a consolidação da construção do mesmo, como também para sua aceitação por parte dos construtores de software e sua extensão. Com isso, o atual trabalho tem o objetivo de apresentar conceitos arquiteturais atuais e praticáveis, a fim de tornar a arquitetura proposta bem fundamentada e com capacidades facilitadas de integração e extensão.

O foco deste trabalho é o modelo de referência e os componentes arquiteturais derivados deste modelo. Não é objetivo deste trabalho se aprofundar em áreas como especialidades de negócio de inteligência artificial para mineração, entre outras. Busca-se com os resultados deste trabalho oferecer um ambiente onde inúmeros componentes possam ser integrados de uma maneira facilitada, para fazer parte do processo de descoberta de conhecimento.

1.1.1. Objetivos específicos

Este trabalho não poderia ficar somente no projeto estrutural, ou ainda, na implementação de um protótipo. Com isto, buscou-se na literatura subsídios que pudessem fornecer as melhores práticas arquiteturais e de desenvolvimento de software para a construção de um *framework* estável que vai além de um simples protótipo para validação deste trabalho. Com isto, foram definidos alguns objetivos específicos:

- Definição de um modelo de referência para servir de guia na especificação das funcionalidades do software.
- Definição de uma arquitetura de referência para abordar mais diretamente os problemas não funcionais definidos no modelo de referência, fornecendo uma base de componentes para a construção da arquitetura de software, que por sua vez é facilitada e concentrada nos requisitos funcionais do modelo de referência.
- Desenvolvimento de um *framework* utilizando a tecnologia Java® para testar e validar a arquitetura de referência proposta e as soluções por ela definidas.
- Construção de um sistema KDD que utiliza a arquitetura de software definida e o *framework* desenvolvido.

Este trabalho também busca unir a área de descoberta de conhecimento (muito preocupada com o processo KDD e esquecendo questões de qualidade e organização do software, focando muito o resultado imediato) com a área de engenharia de software.

1.2. Contribuições

Como principais contribuições deste trabalho tem-se:

- Um modelo de referência para sistemas KDD que define os principais módulos e o fluxo de dados entre estes módulos.
- Uma arquitetura de referência que resolve diversos problemas arquiteturais que podem ser reusados por outros domínios de aplicação, como por exemplo, controle transacional, controle de segurança, instanciação e comunicação entre os objetos instanciados pela arquitetura.
- Um exemplo prático que mostra desde a concepção do modelo de referência da aplicação até a implementação dos pacotes de software, abordando cada fase da construção do software e a documentação da arquitetura.
- Para se ter um ambiente dinâmico, e construir componentes que sejam capazes de realizar suas tarefas de forma conjunta e seqüencial, foi analisada a importância da utilização de metadados que controlam as atividades de preparação de dados e o acesso ao DW. Com isto, foi definido e implementado módulo para gerenciamento das entidades do sistema KDD, o módulo CRUD. Este módulo também pode ser facilmente reutilizado em outros domínios.

1.3. Desafios

A arquitetura de um software pode influenciar diretamente o seu sucesso ou o seu fracasso. O *Unified Process* (Jacobson e Booch, 1998), que reúne as melhores práticas de desenvolvimento de software e a experiência de longos anos de profissionais da área, apresenta em sua especificação o princípio de processo de software centrado na arquitetura. Os autores do processo reconhecem e comprovam em seus trabalhos, que um desenvolvimento guiado por uma arquitetura bem definida pode resultar em um software de maior qualidade.

É um grande desafio para qualquer desenvolvedor de software propor uma solução que atenda os atuais requisitos apresentados e ao mesmo tempo se prepara para suportar futuros requisitos que poderão ser importantes para o sucesso da organização que utiliza o software.

Um outro desafio deste trabalho foi o de estar documentando a concretização das idéias da arquitetura em um software funcional, bem fundamentado, com estruturas bem definidas, com aplicação de princípios de qualidade e implementando este software em um ambiente de desenvolvimento capaz de integrar uma equipe de programadores. Desta forma, é mostrada a arquitetura desde de a sua concepção inicial com os primeiros diagramas até o software implementado.

1.4. Organização

Além deste capítulo de introdução, este trabalho está dividido em mais três capítulos.

No Capítulo 2 é apresentada uma fundamentação teórica sobre as tecnologias envolvidas neste trabalho e são descritos alguns trabalhos que formaram a base para o desenvolvimento deste trabalho.

No Capítulo 3 é descrito detalhadamente o processo de concepção da arquitetura proposta: definição do modelo de referência e levantamento dos requisitos não funcionais; definição da arquitetura de referência e a definição da arquitetura de software.

No Capítulo 4 são discutidos os detalhes do *framework* que implementa a arquitetura de referência.

No Capítulo 5 é mostrado o uso do *framework* implementado através de um estudo de caso.

Finalmente, no Capítulo 6, são apresentadas uma conclusão e sugestões para trabalhos futuros.

2. – Fundamentação teórica

Para o desenvolvimento deste trabalho foram definidos dois focos de pesquisa. O primeiro concentrado no domínio de sistemas KDD para o levantamento dos requisitos funcionais e não funcionais destes sistemas. O segundo centrado na busca de informações sobre como desenvolver uma arquitetura e torná-la implementável.

Neste capítulo são apresentados os principais conceitos que foram levantados por estes dois focos de pesquisa. Primeiramente são abordados os conceitos envolvidos em ambientes de descoberta de conhecimento. Logo depois, são apresentados alguns conceitos sobre arquitetura de software e desenvolvimento de software e, por fim, são descritos sucintamente alguns trabalhos relacionados a este trabalho.

2.1. Ambientes de Descoberta de Conhecimento

Primeiramente, é importante diferenciar sistemas KDD e sistemas baseados em conhecimento (SBC). Durante as pesquisas do atual trabalho, vários artigos foram encontrados sobre estes dois temas e muitas vezes os assuntos podem se confundir. Sistemas KDD reúnem um conjunto de atividades e ferramentas com o objetivo maior de buscar informações em banco de dados e transformá-las em conhecimento interessante. Isto se dá com a aplicação de consultas específicas para buscar determinado conhecimento sobre um conjunto de dados, ou ainda, com o uso de técnicas de mineração que são aplicadas sobre um conjunto de dados. Os resultados da aplicação dessas técnicas podem ser analisados e interpretados gerando conhecimento útil. A caracterização de um conhecimento útil se faz necessária para separar conhecimentos que não são interessantes do ponto de vista estratégico, operacional, ou de qualquer outra forma para os analistas de negócio (Dias, 2001).

SBC são sistemas que agem com o meio exterior e reúnem informações sobre estas ações de forma a acumular conhecimento e melhorar a integração do sistema com seu meio. Esses sistemas utilizam, dentre diversas outras técnicas, tecnologia de inteligência artificial. Muitas vezes, estes sistemas utilizam banco de dados para o armazenamento do conhecimento adquirido (Ullman, 1998). É aí que se encontra a principal diferença entre esses sistemas: um utiliza o banco de dados para buscar conhecimentos implícitos e o outro utiliza o banco de dados para armazenar conhecimentos adquiridos durante sua interação com ambiente.

2.1.1. Processo de Descoberta de Conhecimento

O processo de descoberta de conhecimento em banco de dados (processo KDD) envolve diversas etapas e ferramentas de suporte para que as etapas possam ser satisfatoriamente cumpridas. Na literatura são encontradas diversas propostas de processo KDD que definem etapas a serem seguidas na busca de conhecimento em banco de dados. A seguir são apresentadas algumas dessas propostas.

Processo KDD segundo Dias

Tomando como base as propostas de processos KDD de Feldens (1998 apud Dias, 2001), Groth (1998 apud Dias, 2001) e Lans (1997 apud Dias, 2001), Dias (2001) propôs a divisão do processo KDD em seis passos básicos, conforme é mostrado na Figura 2.1. A seguir, esses passos são descritos sucintamente.

Na **Definição de Objetivos**, devem ser definidos os objetivos de negócio que deverão ser alcançados com a mineração de dados e o que deverá ser feito com os seus resultados, como por exemplo: mudança de plano de marketing.

A **preparação de dados** envolve as tarefas de seleção e transformação dos dados. Os dados selecionados e transformados são armazenados em um DW, *data mart*[1] ou *data set*[2]. Para facilitar a realização desta fase, deve ser mantido um catálogo de metadados sobre as fontes de dados e sobre o que está no DW, *data mart* e *data set*. A realização das tarefas deste passo exige conhecimento dos dados operacionais e de seus relacionamentos, disponibilidade de tempo do analista e/ou usuário e alguns cuidados na escolha de subconjuntos de atributos e de dados.

A **Definição de um estudo** envolve a articulação de um alvo esperado ao final da execução do processo KDD. Este alvo influencia a escolha de uma variável dependente ou uma saída que caracterize um aspecto do alvo e também, a especificação dos itens de dados que deverão ser utilizados no estudo.

---

[1] Um *data mart* é um DW departamental, ou seja, um DW construído para um departamento específica da organização. Desta forma, é diminuído o impacto e os riscos da implantação de um DW. (Inmon, 1997).

[2] Um *data set* é um conjunto de dados, geralmente, armazenado em estruturas relacionais de tabelas.

A **construção de um modelo** é feita através de uma técnica de mineração de dados, tendo como base os dados transformados e o estudo definido no passo anterior.

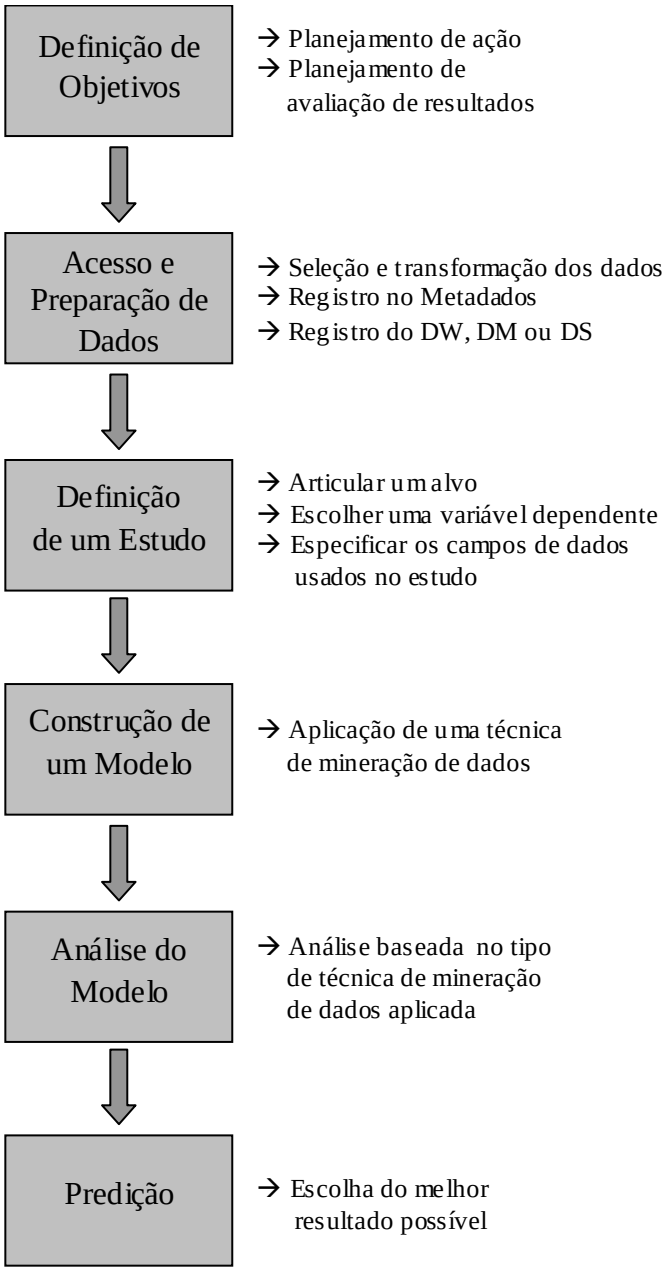


Figura 2.1: Processo de descoberta de conhecimento. Fonte Dias (2001).

A **análise do modelo** depende do tipo de modelo usado para representar os dados, conforme descrito no passo anterior. Cada modelo gera resultados em diferentes formatos, e a interpretação destes resultados é diretamente ligada à técnica de mineração proposta pelo modelo.

A **predição** é o processo de escolher o melhor resultado possível baseado na análise de dados históricos. O usuário deve analisar a informação descoberta de acordo com sua tarefa de suporte à decisão e objetivos. Portanto, ele precisa ter um bom entendimento sobre o negócio da empresa e sobre o conhecimento descoberto.

Processo KDD segundo Han e Kamber

Han e Kamber (2001) definem um processo KDD com sete passos mostrados na Figura 2.3: limpeza, integração, seleção, transformação, mineração de dados, avaliação e apresentação. Opcionalmente pode ser construído um Data Warehouse como uma área de integração de dados.

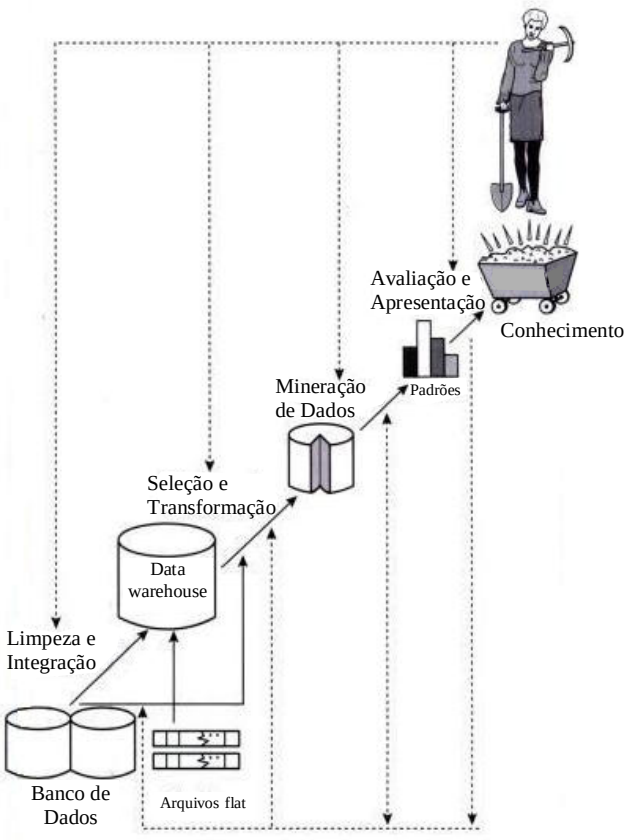


Figura 2.2: Processo KDD com sete passos proposto por Han e Kamber (2001).

Processo KDD segundo Fayyad

As principais caracterizações do processo KDD, por Fayyad (1996) são: (1) a natureza interativa e iterativa, (2) a condição para obter conhecimento possui nove passos básicos que são mostrados na Figura 2.3. Interativo porque a busca torna-se inválida e inadequada se não houver mecanismos de comunicação com o usuário, principalmente no domínio da aplicação,

e a avaliação de padrões interessantes. Iterativo porque possibilita repetições entre quaisquer dois dos nove passos do processo KDD.

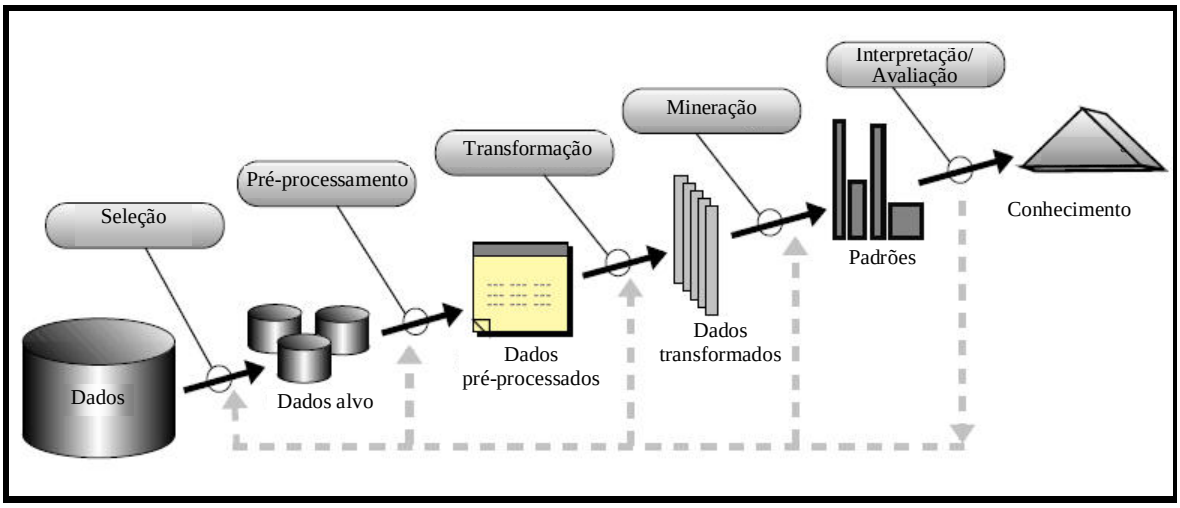


Figura 2.3: Processo KDD com nove passos proposto por Fayyad (1996).

2.2. Data Warehouse

2.2.1. Histórico

A evolução dos sistemas de apoio à decisão pode ser dividida em cinco fases entre 1960 e 1980. No início da década de 1960 o mundo da computação consistia na criação de aplicações individuais que eram executadas sobre arquivos mestres, caracterizadas por programas e relatórios.

Aproximadamente em 1965, o crescimento dos arquivos mestres e das fitas magnéticas explodiu, surgindo problemas como: a complexidade de manutenção dos programas; a complexidade do desenvolvimento de novos programas; a quantidade de hardware para manter todos os arquivos mestres e a necessidade de sincronizar dados a serem atualizados.

Por volta de 1970, surgiu a tecnologia DASD (*direct access storage device*), substituindo as fitas magnéticas pelo armazenamento em disco. Com o DASD surgiu um novo tipo de software conhecido como SGBD ou Sistema de Gerenciamento de Banco de Dados, que tinha o objetivo de tornar o armazenamento e o acesso a dados no DASD mais fáceis para o programador. E com o SGBD surgiu a idéia de um “banco de dados” que foi definido como: uma única fonte de dados para todo o processamento.

Aproximadamente em 1975 surgiu o processamento de transações *online*. Com o processamento de transações *online* de alto desempenho, o computador pôde ser usado para tarefas que antes não eram viáveis, como: controlar sistemas de reservas, sistemas de caixas bancários, sistemas de controle de produção e outros.

Até o início da década de 1980, novas tecnologias, como os PCs (*personal computers*) e as L4Gs (linguagens de quarta geração), começaram a aparecer. O usuário final passou a controlar diretamente os sistemas e os dados, descobrindo que era possível utilizar os dados para outros objetivos além de atender ao processamento de transações *online* de alto desempenho. Foi nesse período também que se tornou viável a construção dos MIS (*management information systems*), hoje conhecidos como SAD (Sistema de Apoio à Decisão). Eles consistiam em processamento utilizado para direcionar decisões gerenciais. O DW surgiu como um agrupamento e uma formalização de diversos conceitos existentes. Atualmente, uma área em plena evolução.

Inmon (1997), considerado um pioneiro no tema, apresenta a definição de um DW como sendo uma coleção de dados orientada por assuntos, integrada, variante no tempo e não volátil, que tem por objetivo dar suporte aos processos de tomada de decisão, possibilitando que gerentes, executivos e analistas tomem decisões melhores e mais rapidamente.

O conceito e os benefícios de um DW têm se tornado conhecidos pelas empresas que estão cada vez mais direcionando investimentos para área de TI.

2.2.2. Características

A seguir está uma breve descrição das características de DW, conforme (Inmon, 1997).

Orientação por Assunto: Significa que um DW armazena as informações agrupadas por assuntos de interesse da empresa que são mais importantes, em contraste com os sistemas operacionais que são orientados a processos desenvolvidos para manter as transações realizadas diariamente (Machado, 2000).

Variação de Tempo: Os dados de um DW representam resultados operacionais em um determinado momento de tempo, o momento em que foram capturados. Os dados de um DW são um *snapshot*, um conjunto estático de registro de uma ou mais tabelas, capturados em um

momento de tempo predeterminado. Isto significa que os dados de um DW não podem ser atualizados (Machado, 2000).

Não Volátil: O DW possui duas operações básicas: inclusão de dados e acesso a esses dados em modo de leitura. Os dados que são incluídos no DW provêm de um ambiente operacional e, após sofrerem as modificações necessárias, estarão disponíveis para a análise de tomada de decisão. Esses dados continuam fazendo parte do DW até que eles se tornem obsoletos ou irrelevantes, após isso, novos dados são incluídos no DW e analisados através de acesso em modo de leitura.

Integração de dados: A integração de dados é uma das principais características do DW pelo fato das representações dos dados de diversos sistemas serem padronizadas em uma única representação. Podem existir diversas formas de representar uma mesma estrutura de dados, e para manter a consistência desses dados, uma única forma de representação deve ser disposta. Um exemplo deste problema seria a representação do sexo, em uma definição pode-se ter como forma de representar o sexo como um campo caractere “M” para masculino e “F” para feminino, e em outra pode-se representar como um campo numérico “1” para masculino e “2” para feminino. Para que estes dados possam ser utilizados no DW, seria necessário realizar uma integração, através de um filtro, que definiria um formato padronizado para ser utilizado no DW.

2.2.3. Arquiteturas de DW

A arquitetura de DW é um conjunto de documentos, planos, modelos, projetos e especificações, com seções separadas para cada área de componente chave e detalhes suficientes para permitir sua implementação (Hardley, 2005).

De acordo com Hadley (2005), uma arquitetura de DW inclui dados e elementos técnicos e pode ser quebrada em três áreas: arquitetura de dados, infra-estrutura e área técnica. A arquitetura de dados está centrada nos processos de negócio. A área de infra-estrutura inclui hardware, rede, sistemas de operação e máquinas *desktop*. A área técnica compreende tecnologias de tomada de decisão que serão necessárias para os usuários, bem como suas estruturas de suporte.

A Figura 2.4 mostra um modelo lógico que fornece os elementos de alto nível da arquitetura técnica. Segundo Kimball et al (1998), a representação física provavelmente será muito

diferente, depende de muitos fatores, incluindo a maturidade do DW, seu tamanho e a natureza.

A arquitetura técnica tem dois tipos principais de componentes: serviços e depósitos de dados. Serviços são as funções necessárias para realizar as tarefas requeridas no DW. Por exemplo, copiar uma tabela de um lugar a outro é um tipo de serviço básico de movimentação de dados. Depósitos de dados são os lugares temporários ou permanentes dos dados.

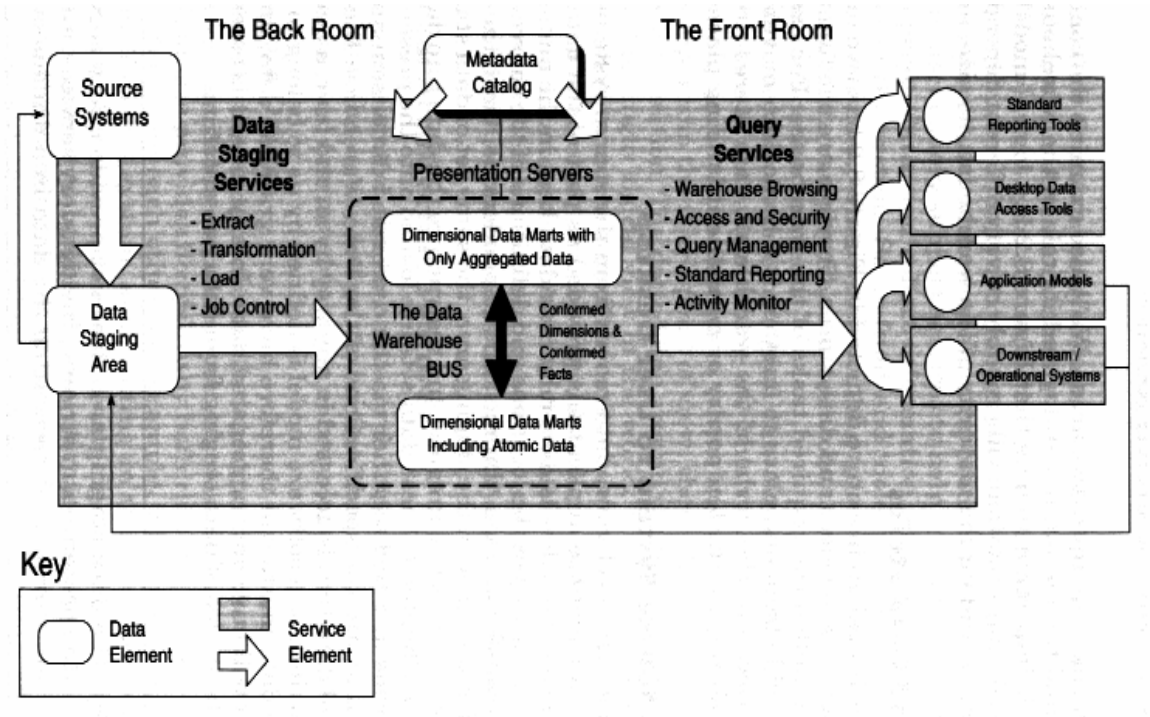


Figura 2.4: Modelo de Arquitetura Técnica de DW (Kimball et al, 1998).

Singh (2001) propôs um modelo de arquitetura em múltiplas camadas para DW, como mostra a Figura 2.5.

Nesta arquitetura, a Camada de Mensagem da Aplicação tem como responsabilidade o transporte das informações do DW para o resto da empresa.

A Camada de Metadados tem como objetivo o acesso aos dados do DW pelos usuários de forma única, sem que eles necessitem saber onde estão ou quais são os formatos dos dados.

A Camada de Acesso à Informação possibilita o acesso às informações do *DW* através de relatórios, ferramentas OLAP e mineração de dados, permitindo que o usuário manipule as informações do DW.

A Camada de Acesso a Dados é responsável pela conexão entre a camada de dados operacionais e a camada de estágio de dados, realizando esta conexão de modo transparente para o usuário.

A Camada *DW* é onde estão armazenados os dados. Normalmente é uma base utilizando a modelagem multidimensional, para permitir um acesso rápido e flexível.

Na Camada de Estágio dos Dados são realizadas a extração, a limpeza e a transformação dos dados antes para que eles possam ser carregados no DW.

Na Camada de Dados Operacionais estão contidas as bases de dados da empresa ou organização.

A Camada de Gerenciamento do Processo gerencia os diversos processos que estão envolvidos em um *DW*.

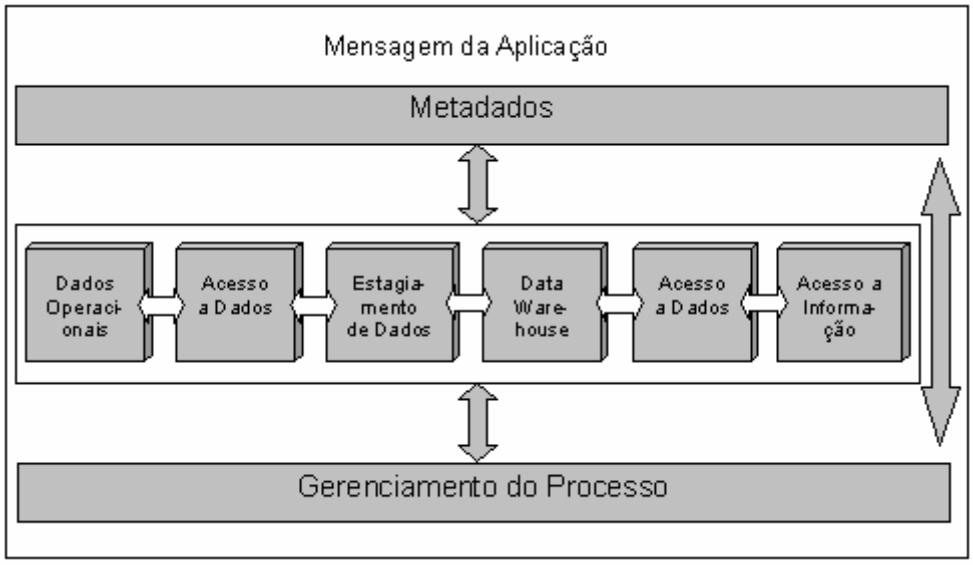


Figura 2.5: Arquitetura proposta por Singh (2001).

Sell (2001) definiu um modelo em que a principal preocupação é uma arquitetura que seja flexível o suficiente para integrar fontes de dados distintas e permitir alterações nas fontes ou incorporações de novas fontes se necessário. A Figura 2.6 mostra este modelo. Para cada fonte é desenvolvido um módulo de coleta que conhece o formato dos dados da fonte, obtém os dados e armazena-os na Área de Estágio. O módulo Transformação analisa os dados da Área de Estágio e realiza as transformações necessárias para que os dados fiquem homogêneos e possam ser carregados para os *Data Marts* pelo módulo Incorporação. Os dados do DW são disponibilizados por uma aplicação em um servidor de aplicação.

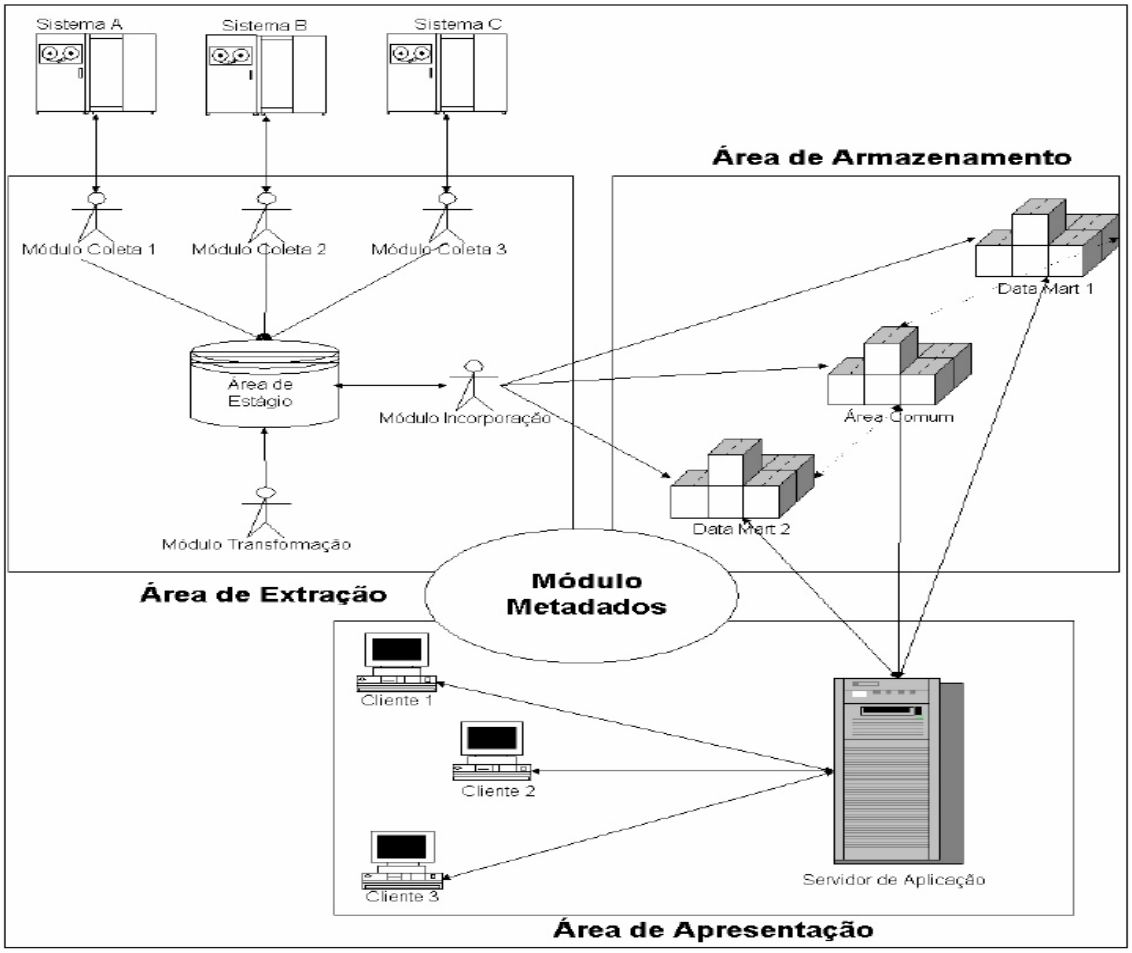


Figura 2.6: Arquitetura proposta por Sell (2001).

A arquitetura de DW proposta por Menolli (2004) é uma arquitetura em camadas de *data marts* integrados e incrementais, conforme mostra a Figura 2.7.

A camada ETL é dividida em dois módulos, um módulo de migração e outro de extração, transformação e carga. O módulo de migração transforma dados em diversos formatos para um formato referência, que é o formato escolhido pelo projetista de DW para ser o formato do DW. O módulo de extração, transformação e carga coleta dados relevantes, que são previamente definidos na fase de planejamento e definição dos requisitos, e os armazena na área de estágio, transformados e limpos.

Posteriormente, na camada Área de Estágio, os dados são integrados e incorporados aos *data marts.* As regras de extração, transformação e integração são registradas no repositório de metadados.

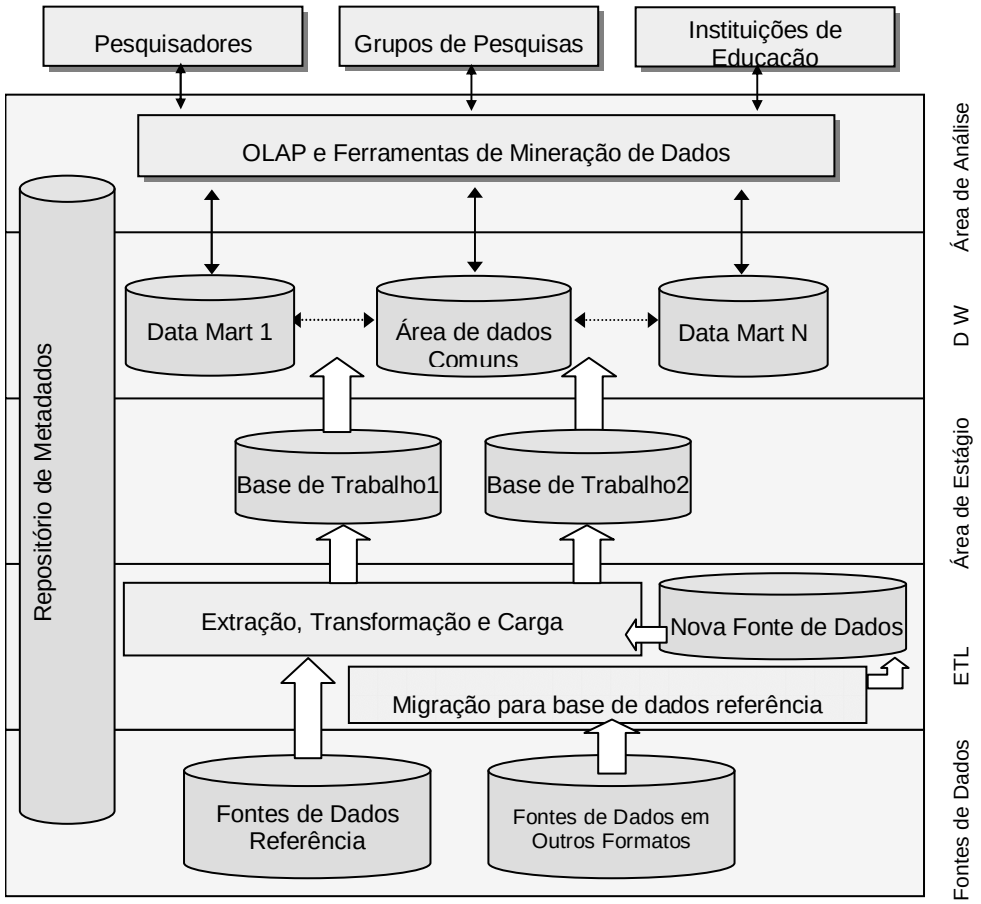


Figura 2.7: Arquitetura proposta por Menolli (2004).

2.3. Sistemas OLTP e Sistemas OLAP

Em uma organização podem-se ter diversas fontes (provedores) de dados que podem estar em um único local ou distribuídos. Essas fontes são manipuladas por vários sistemas que controlam as rotinas operacionais da organização. Além do uso dessas fontes nas rotinas operacionais, essas fontes podem ser usadas para extração de informações úteis para tomada de decisão. A Figura 2.8 ilustra a divisão de uma organização em dois grandes departamentos distintos caracterizados pelo tipo de sistemas que utilizam, o departamento operacional que utiliza sistemas OLTP e o departamento gerencial que utiliza sistemas OLAP:

- Departamento operacional: utiliza sistemas de controle, faturamento, produção, comunicação, registros, entre outros. Todos estes sistemas executam inúmeras operações e acessam dados de diversas fontes. As operações destes sistemas são basicamente estruturadas em transações, daí a classificação destes como OLTP (*On-line transactional processing*). Sendo estas operações otimizadas para obter maior

desempenho na execução de transações, bem como, apresentar recursos de tratamento de concorrências e tolerância a falhas (Chaudhuri; Dayal, 2004).

- Departamento gerencial: este departamento se preocupa com a saúde da organização, necessitando de relatórios e informações para apoiar o processo de decisão. Utiliza sistemas que analisam grandes quantidades de dados e geram gráficos, planilhas entre outros recursos, que transformam os dados em informações. Estes sistemas são classificados como OLAP (*On-line analytical processing*), pois são otimizados para ambientes onde o tempo de resposta das pesquisas e o volume de dados são fatores determinantes (Chaudhuri; Dayal, 2004).

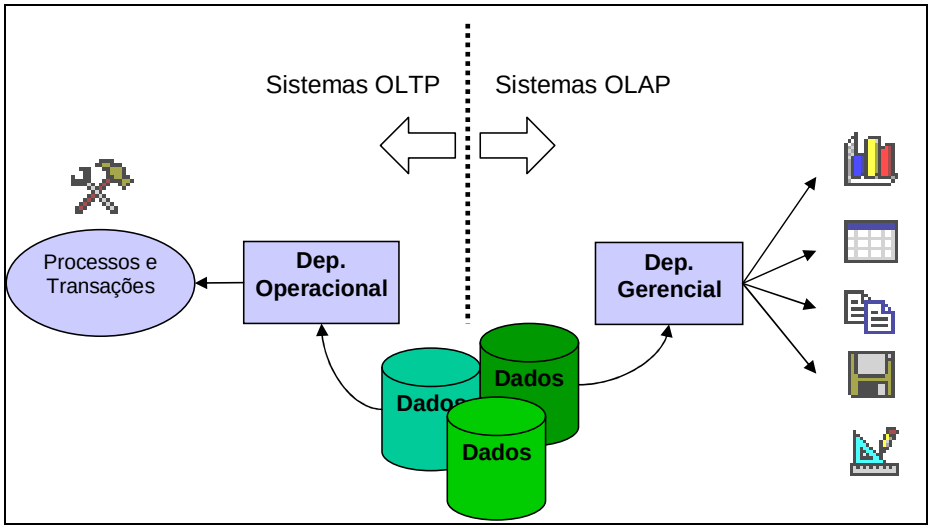


Figura 2.8: Divisão de uma organização em duas visões departamentais: Operacional e Gerencial.

2.4. Mineração de dados

Segundo Goebel e Gruenwald (1999 apud Dias, 2001), a mineração de dados pode ser considerada como uma parte do processo de Descoberta de Conhecimento em Banco de Dados (*KDD – Knowledge Discovery in Databases*). O termo KDD é usado para representar o processo de tornar dados de baixo nível em conhecimento de alto nível, enquanto mineração de dados pode ser definida como a extração de padrões ou modelos de dados observados.

A mineração de dados combina métodos e ferramentas das seguintes áreas: aprendizagem de máquina, estatística, banco de dados, sistemas especialistas e visualização de dados, conforme Figura 2.9.

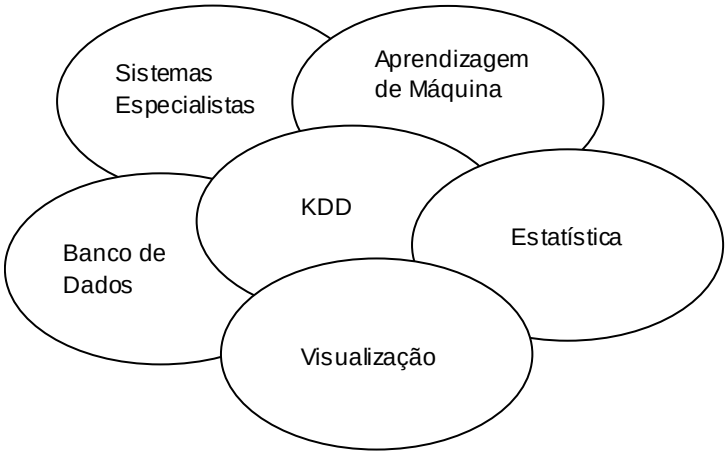


Figura 2.9: A mineração de dados como um campo multidisciplinar. Fonte Cratochvil (1999 apud Dias, 2001).

Berry e Linoff (1997 apud Dias, 2001) definem a mineração de dados como sendo a exploração e a análise, por meio automático ou semi-automático, de grandes quantidades de dados, a fim de descobrir padrões e regras significativos.

Os principais objetivos da mineração de dados são descobrir relacionamentos entre dados e fornecer subsídios para que possa ser feita uma previsão de tendências futuras baseada no passado. Os resultados obtidos com a mineração de dados podem ser usados no gerenciamento de informação, processamento de pedidos de informação, tomada de decisão, controle de processo e muitas outras aplicações.

A Tabela 2.1 apresenta de forma resumida as principais técnicas de mineração de dados e as tarefas[1] por elas realizadas.

[1] Tarefa é um tipo de problema de descoberta de conhecimento a ser solucionado Dias (2001).

Tabela 2.1: Técnicas de Mineração de Dados. Fonte (Dias, 2001), adaptada pelo autor.

TÉCNICA	DESCRIÇÃO	TAREFAS
Descoberta de Regras de Associação	Estabelece uma correlação estatística entre atributos de dados e conjuntos de dados	• Associação
Árvores de Decisão	Hierarquização dos dados, baseada em estágios de decisão (nós) e na separação de classes e subconjuntos	• Classificação • Regressão
Raciocínio Baseado em Casos ou MBR	Baseado no método do vizinho mais próximo, combina e compara atributos para estabelecer hierarquia de semelhança	• Classificação • Segmentação
Algoritmos Genéticos	Métodos gerais de busca e otimização, inspirados na Teoria da Evolução, onde a cada nova geração, soluções melhores têm mais chance de ter "descendentes"	• Classificação • Segmentação
Redes Neurais Artificiais	Modelos inspirados na fisiologia do cérebro, onde o conhecimento é fruto do mapa das conexões neuronais e dos pesos dessas conexões	• Classificação • Segmentação

2.5. Arquitetura de Software

A informática é uma área bastante nova comparada com outras áreas da ciência. Passamos por um desenvolvimento baseado em experiências pessoais até o surgimento de modelos arquiteturais, padrões de projetos e metodologias de desenvolvimento de software. A utilização de computadores e, conseqüentemente, de software foi crescendo de tal forma que a maneira pela qual o software era desenvolvido começou a ganhar maior atenção.

Tait (2000) afirma que até os anos 80 a arquitetura somente era associada com projetos de hardware. Por volta de 1987 a arquitetura passou a ser associada à área de software. Os engenheiros começaram a evoluir o desenho de abstração do seu software a fim de melhor compreendê-lo. Já nos anos 90 a arquitetura começou a ser utilizada para expressar sistemas de informações que fazem parte da estratégia de negócio das instituições. Sistemas mais complexos começaram a ser realidade e, no final dos anos 90, os conceitos de arquitetura começaram, também, a ser usados na definição da arquitetura de informação da empresa.

Bass et al. (2003) definem arquitetura de software como sendo a estrutura ou estruturas de sistema que compreendem os elementos de software, as propriedades externamente visíveis destes elementos e os relacionamentos entre eles. Outra definição dada por Shaw e Garlan (1996) é que uma arquitetura de software descreve os elementos a partir dos quais o sistema é construído, as interações entre esses elementos, padrões que guiam sua composição e as restrições desses padrões. Ou seja, um sistema é definido a partir de uma coleção de componentes e da especificação de como esses componentes interagem entre si para a

realização dos papéis assumidos pelo sistema, sendo tudo isto representado por sua arquitetura.

A arquitetura do software forma a espinha dorsal de todo o sistema de software bem sucedido. Uma arquitetura é o portador preliminar dos atributos de qualidade de sistema de software tais como o desempenho ou a confiabilidade. Uma arquitetura correta, projetada para atender os atributos de qualidade requeridos, claramente documentada e bem avaliada é o caminho para o sucesso do projeto do software, caso contrário, é uma receita para o desastre garantido.

A seguir são apresentados os principais conceitos sobre arquitetura de software que são importantes para a atual proposta. Tais como estilos arquiteturais e aplicação de padrões no desenvolvimento da arquitetura de uma aplicação.

2.5.1. Padrões Arquiteturais

Padrões são soluções de qualidade para problemas que são comumente vistos em determinados contextos. Exemplos de padrão são o MVC e o SOA, que apresentam soluções para organizar componentes de software. Estes padrões são descritos nas próximas seções..

A utilização de padrões na construção de um software pode significar um grande ganho na corretude do sofware e, conseqüentemente, na sua qualidade. Além disso, o entendimento do software por parte dos arquitetos e programadores poderá se tornar mais claro, o que vem a acrescentar um grande benefício no desenvolvimento do projeto de software (Broemmer, 2003).

Padrões não se desenvolvem, são identificados. Dentre as categorias de padrões definidos na literatura (Coplien, 1995), (Gamma 1995), (Vlissides, 1996), (Buschmann, 1996), (Martin, 1998), e (Metsker, 2002) têm-se padrões arquiteturais que expressam o esquema ou organização estrutural fundamental de sistemas de software ou hardware.

Bass et al. (2003) apresenta o relacionamento entre padrões para a especificação de uma arquitetura.(Figura 2.10).

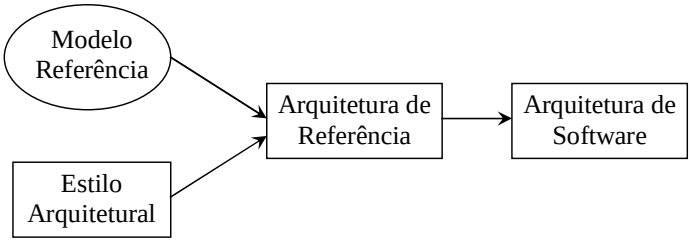


Figura 2.10: Relacionamento entre os padrões para a composição de uma arquitetura de software.

Modelo de Referência

Segundo OASIS (2006), um modelo de referência é uma estrutura abstrata utilizada para representar os relacionamentos significativos entre as entidades de um determinado ambiente. Este modelo serve como base para o desenvolvimento da arquitetura de referência, ou mesmo, da arquitetura de software diretamente. Um modelo de referência consiste em um conjunto mínimo de conceitos, de axiomas e de relacionamentos pertencentes a um domínio em particular, sendo independente de padrões, de tecnologias, de implementação ou de quaisquer outros detalhes mais concretos.

De acordo com Merson (2006), para sua composição ainda não foi definida nenhuma representação formal. Blocos e linhas podem ser utilizados, seguidos de uma descrição textual sobre a funcionalidade de cada elemento apresentado.

Estilo arquitetural

Algumas coleções de componentes e padrões de interação são comuns a diversos domínios de aplicação e foram agrupados em estilos arquiteturais.

Um estilo arquitetural é uma organização estrutural padrão que pode ser aplicada em diferentes sistemas. Cada estilo possui suas aplicações específicas, auxiliando na solução de um determinado problema. Durante a composição de uma arquitetura de um grande sistema, podem ser apresentadas partes menores que internamente possuem seus próprios estilos arquiteturas.

A seguir são relacionados os estilos mais encontrados na literatura. Somente uma descrição breve é apresentada, maiores detalhes e diagramas representativos podem ser analisados em (Buschmann, 1996).

- **Canos e filtros (*Pipe and filters*):** Arquitetura indicada para problemas que seguem uma seqüência bem definida para serem solucionados, precisando de diversas transformações (filtros). O exemplo mais comum é um compilador de linguagem de programação que tendo um código fonte como entrada, efetua análises léxica, sintática e semântica, gera códigos intermediários, faz ligação com códigos pré-compilados e gera na saída um arquivo executável.
- **Camadas (*Layers*):** O sistema é divido em camadas e cada camada provê serviços para a camada superior. Uma camada superior somente conhece e utiliza uma camada imediatamente inferior. Alterações internas nas camadas não comprometem o funcionamento das demais camadas do sistema.
- **Quadro-negro (*Black-board)*:** Este estilo apresenta um repositório central de dados e diversos componentes que acessam este repositório e realizam uma transformação específica. Ao final de inúmeras transformações, as informações iniciais são transformadas numa solução aceita do problema determinado. Este estilo é utilizado em sistemas de reconhecimento de voz. Uma entrada inicial de ondas sonoras é analisada por diversos componentes e cada componente gera um resultado que é concatenado e analisado novamente para se reconhecer uma voz.

Arquitetura de referência

Segundo Gallagher (2002), uma arquitetura de referência é a generalização da arquitetura de vários sistemas que compartilham um ou mais domínio comum. A arquitetura de referência define a infra-estrutura comum para os sistemas e as interfaces dos principais componentes estruturais que serão incluídos nestes sistemas.

Uma arquitetura de referência é instanciada para criar uma arquitetura de software para um sistema específico. A definição de uma arquitetura de referência facilita a derivação e a extensão de novas arquiteturas de software para classes de sistemas comuns. Uma arquitetura de referência, contudo, desempenha dois papéis importantes sobre os sistemas alvos. Primeiro, ela generaliza e extrai funcionalidades e configurações comuns. Segundo, ela provê uma base sólida para o desenvolvimento de sistemas e um menor custo de desenvolvimento pelo reuso.

Bass et al. (2000) afirma que o objetivo da especificação de uma arquitetura de referência é criar um ambiente de desenvolvimento que possa ser usado para construir aplicações de forma mais rápida e com melhor desempenho, qualidade e reusabilidade.

2.5.2. SOA (*Service Oriented Architecture*)

Uma arquitetura orientada a serviço define que as regras de negócio de um domínio podem ser abstraídas em serviços. O'Brien et al (2005) apresentam os principais princípios relacionados a este paradigma:

Autonomia: um serviço é independente de outros serviços para execução de suas atividades. Ele possui um controle completo de sua execução dentro da fronteira por ele ocupada.

Baixo acoplamento: um serviço deve ser projetado para interagir sobre uma base com o mínimo de acoplamento possível.

Abstração: a única parte que é visível pelo ambiente externo do serviço é sua interface, que muitas vezes é exposta por uma descrição e contrato formal. A lógica interna e o comportamento do serviço são invisíveis e irrelevantes para quem os requisita.

Contrato formal: um serviço deve especificar um contrato formal que define os termos de troca de informações e qualquer outra informação suplementar sobre o serviço.

Não há uma definição oficial para SOA. Em conseqüência, SOA é definido em muitas maneiras diferentes, incluindo:

- "Uma arquitetura orientada a serviço é um *framework* que quebra a aplicação de negócio em funções individuais e processos de negócio, chamados serviços. Uma SOA permite construir, implantar e integrar esses serviços, independente da aplicação ou plataforma de software sobre a qual são executados" - *IBM Corporation*.
- "É uma proposta para organizar a tecnologia de informação na qual os recursos de dados, lógica e infra-estrutura são acessados por mensagens entre rede e interfaces" – *Microsoft*.

- "Uma SOA é um conjunto de componentes que podem ser invocados, e cujas descrições da relação podem ser publicadas e descobertas". – *Consortium World Wide Web*[1].

Cada serviço é responsável por uma função do negócio, e toda a aplicação que necessitar executar esta função usa este serviço. Além disso, uma aplicação é criada montando e coordenando as atividades entre os serviços apropriados para compor o processo de negócio. Os serviços dentro da aplicação possuem baixo acoplamento e são reusáveis através das várias aplicações criadas. Uma aplicação não precisa ser necessariamente composta somente de serviços, mas pode ser.

Cherobino (2006) destaca que SOA apresenta um curto tempo para a entrega de novas funcionalidades. Além disso, qualquer mudança no ambiente de negócios, de um lançamento de produto concorrente a uma fusão entre companhias, tem resposta mais rápida. Com SOA, os processos podem ser alterados rapidamente já que os serviços são disponibilizados pelos aplicativos e sistemas, tanto os adquiridos no mercado quanto aqueles desenvolvidos internamente, para serem utilizados por qualquer novo sistema ou processo que os necessite, evitando a redundância..

Com a consolidação da cultura de SOA, é possível otimizar o orçamento disponível para TI. A atual "torre de Babel" - com inúmeros fornecedores compondo a mesma infra-estrutura e tendo de compartilhar informações em padrões distintos – é substituída por um ambiente mais transparente e interoperável, mesmo com soluções de diversos fabricantes. Isto faz cair os custos relacionados à manutenção dos sistemas e às aplicações para integração, abrindo espaço para a aquisição de novas soluções ou tecnologias.

2.5.3. MVC

Model-View-Controller (Modelo-Visão-Controlador) é um padrão de projeto que apresenta uma clara separação da interface do usuário (visão) e a lógica de negócio (modelo). O controlador ajuda a gerenciar as entradas e as saídas entre visão e modelos. As vantagens deste modelo são a facilidade de manutenção, a flexibilidade e a legibilidade do código.

---

[1] Worldwide Web Consortium (W3C). Web Services Glossary. http://www.w3.org/TR/ws-gloss/ (February 2004)

Tate (2002) discute que na correta aplicação deste padrão na Internet, que não é interativa mas um modelo baseado em requisição e resposta, é necessário considerar duas visões: a visão HTML do lado cliente e a visão da sessão do usuário no lado do servidor. Utilizando as definições apresentadas pela autora, foi elaborada a Figura 2.11 que mostra a aplicação do padrão MVC na WEB.

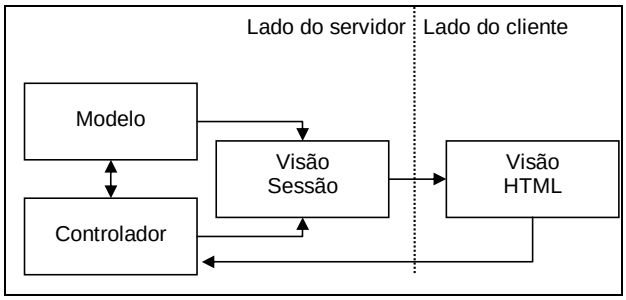


Figura 2.11: Aplicação do modelo MVC na WEB com duas visões.

2.5.4. Framework

Do ponto de vista da estrutura de um *framework*, Coad (1992) o define como um esqueleto de classes, objetos e relacionamentos agrupados para construir aplicações específicas e Johnson et al (1988) o define como um conjunto de classes abstratas e concretas que provê uma infra-estrutura genérica de soluções para um conjunto de problemas. Essas classes podem fazer parte de uma biblioteca de classes ou podem ser específicas da aplicação.

Um dos principais objetivos dos *frameworks* é possibilitar reuso, não só de componentes isolados, como também de toda a arquitetura de um domínio específico e ainda dos esforços.

Uma característica importante dos *frameworks* é que os métodos definidos pelo usuário para especializá-lo são chamados de dentro do próprio *framework*, ao invés de serem chamados do código de aplicação do usuário (Johnson 1988). O *framework* geralmente faz o papel de programa principal, coordenando e seqüenciando as atividades da aplicação. Essa inversão de controle dá força ao *framework* para servir de esqueletos extensíveis. Os métodos fornecidos pelo usuário especializam os algoritmos genéricos definidos no *framework* para uma aplicação específica.

Classificação de Frameworks

Segundo Fayad et al (1997, apud Braga (2002)), os *frameworks* são classificados em três grupos: *frameworks* de infra-estrutura do sistema, *frameworks* de integração de *middleware* e *frameworks* de aplicação empresarial.

Os *frameworks* de infra-estrutura do sistema simplificam o desenvolvimento da infra-estrutura de sistemas portáveis e eficientes, como por exemplo, os sistemas operacionais, sistemas de comunicação, interfaces com o usuário e ferramentas de processamento de linguagem. Em geral são usados internamente em uma organização de software e não são vendidos a clientes diretamente.

Os *frameworks* de integração de *middleware* são usados, em geral, para integrar aplicações e componentes distribuídos. Eles são projetados para melhorar a habilidade de desenvolvedores em modularizar, reutilizar e estender sua infra-estrutura de software para funcionar de forma bastante integrada em um ambiente distribuído. Exemplos dessa classe de *framework* são o "Object Request Broker" (ORB), *middleware* orientado a mensagens e bases de dados transacionais.

Os *frameworks* de aplicação empresarial estão voltados a domínios de aplicação mais amplos e são a pedra fundamental para atividades de negócios das empresas, como por exemplo sistemas de telecomunicações, aviação, manufatura e engenharia financeira. *Frameworks* dessa classe são mais caros para desenvolver ou comprar, mas podem dar um retorno substancial do investimento, já que permitem o desenvolvimento de aplicações e produtos diretamente.

2.6. Trabalhos relacionados

Esta seção descreve alguns trabalhos relacionados ou que oferecem alguma contribuição para o desenvolvimento da pesquisa do atual trabalho. A descrição apresentada é bastante breve, e mais informações podem ser obtidas nas referências indicadas.

2.6.1. Oracle Warehouse Builder

O Oracle Warehouse Builder é uma ferramenta que faz parte do conjunto de soluções para banco de dados da empresa Oracle. Este ambiente permite o acesso a bases de dados, a definição de transformações, a criação de novas bases de dados e à operação com bases de

dados multidimensionais. Possui uma interface gráfica bastante elaborada, com recursos avançados para edição visual do modelo de DW construído. Todos os metadados sobre o modelo são armazenados em um repositório de metadados, que possui um controle de acesso por usuários. Todo o ambiente é integrado com o banco de dados Oracle. O ambiente possui extensões para carga de dados em outras bases de dados não Oracle e arquivos textos (Oracle, 2005).

A proposta de armazenamento dos metadados sobre as estruturas das fontes e dos destinos dos dados, como também sobre todo o processo ETL, serviu como base para o atual trabalho. O armazenamento desses metadados evita re-trabalho do operador do sistema KDD.

2.6.2. Uma Arquitetura para Descoberta de Conhecimento e Gerenciamento de Conhecimento (*Architecture for knowledge Discovery and knowledge management)*

Baseado no modelo de processo KDD proposto por Reinartz (1999, apud Gupta et al, 2004), é definido um ambiente que estende o processo convencional e insere novas características, tais como: suporte para reuso de computação, independência de fonte de dados, processo de descoberta explicitamente indicado no ambiente, suporte para experimentação de conjuntos de dados, entre outras (Gupta, 2004).

Para a execução das atividades do processo KDD é proposta uma linguagem que é utilizada por um compilador específico, presente na arquitetura, que dispara e controla cada atividade. A vantagem de se ter essa linguagem é a obtenção de uma especificação formal do processo KDD que se pretende executar e uma automatização de sua execução. Por outro lado, isto dificulta a realização de algumas atividades mais interativas onde se deseja definir alguns parâmetros e analisar os resultados de maneira mais rápida e menos burocrática.

Toda a arquitetura é dividida em pequenos módulos bastante coesos funcionalmente. No entanto, a arquitetura proposta não apresenta os detalhes de como os seus elementos se comunicam e as restrições de composição da arquitetura.

2.6.3. GridMiner

A ferramenta GridMiner[1] (Brezany et al, 2003) foi construída sobre um *framework* desenvolvido pelo mesmo grupo, que dentre outros assuntos, pesquisa sobre a aplicação de computação paralela no desenvolvimento de sistemas para descoberta de conhecimento. A Figura 2.12 representa os componentes que compõem a ferramenta. Os componentes são classificados em cinco grupos : *Fabric*, *Grid Core*, *GridMiner Base*, *GridMiner Core* e *GridMiner Workflow*.

Esta ferramenta também utiliza uma linguagem que descreve todas as atividades que devem ser executadas durante o processo KDD. Esta linguagem é interpretada pelo componente *GMOrchS* que define quais serviços deverão ser acionados pelo componente *GMDSCE*.

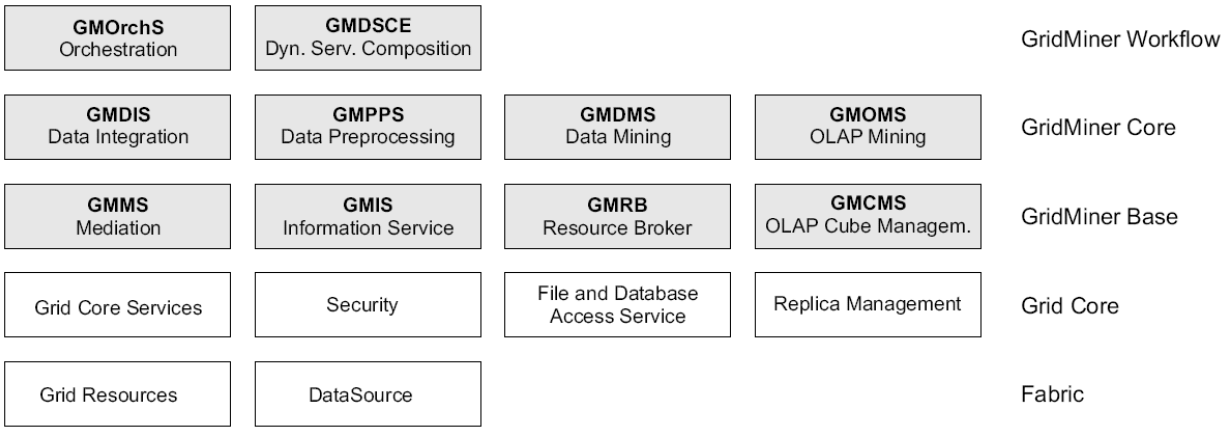


Figura 2.12: Componentes que compõem a ferramenta GridMiner. Fonte: Brezany et al (2003).

2.6.4. KDB2000

O KDB2000 (Appice et al, 2006), cuja arquitetura é mostra Figura 2.13, integra em uma única ferramenta o acesso de base de dados, as técnicas do pré-processamento dos dados e de transformação, algoritmos de mineração de dados e ferramentas de visualização. Esta integração oferece suporte para todo o processo de descoberta de conhecimento em banco de dados. No entanto, a ferramenta não armazena metadados sobre o processo KDD que o operador está executando. Por outro lado, é possível executar algumas atividades e já observar o resultado, tudo em tempo de execução da ferramenta. Desta forma, o processo KDD vai sendo executado interativamente e os resultados parciais de cada atividade podem ser

[1] http://www.gridminer.org

visualizados e, se necessário, as atividades anteriores podem ser retomadas e executadas novamente para correções necessárias.

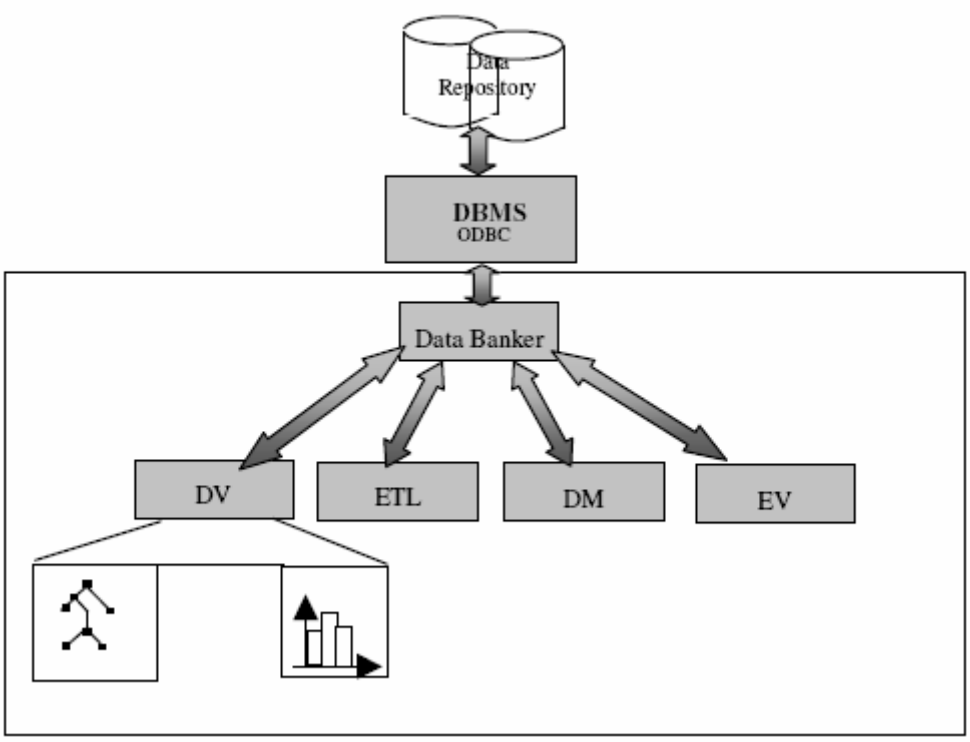


Figura 2.13: Arquitetura da ferramenta KDB2000. Fonte: Appice et al (2006).

O componente *Data Banker* permite o acesso aos dados por alguma fonte externa de dados relacionais usando o controlador de ODBC. São responsabilidades do *Data Banker* conectar a uma base de dados especificada, executar as instruções SQL, e retornar o resultado ao sistema que fez o pedido no formato esperado. Este componente também permite que o sistema solicitante obtenha metadados sobre as tabelas, os itens de dados das tabelas e os seus tipos. Esta informação pode ser usada como parte do processamento dos dados. Conseqüentemente, o *Data Banker* extrai os dados da origem e gerencia o envio destes dados para os demais componentes do KDB2000.

As ferramentas de visualização de dados (DV) são usadas para mostrar os dados selecionados pelo usuário e os padrões descobertos pelos algoritmos de mineração. Este componente é dirigido pela necessidade de visualização dos dados resultantes de qualquer etapa do processo, a fim de permitir uma exploração interativa de grandes volumes de dados.

As técnicas de extração, de transformação e de carregamento (ETL) operam o coração do descobrimento de conhecimento, pois extrai os dados das origens e os transforma para fazê-los acessíveis às atividades de análise.

O módulo de mineração de dados (DM) do KDB2000 inclui algoritmos para a descoberta de padrões nos dados selecionados.

A arquitetura do KDB2000 fornece um módulo de avaliação (EV) dos resultados produzidos pelos algoritmos de mineração de dados.

2.7. Considerações finais

Os conceitos sobre Ambiente de Descoberta de Conhecimento, Data Warehouse, Sistemas OLAP e de Mineração de Dados estão bastante relacionados no que diz respeito à tecnologia da informação. Todos estes conceitos unidos formam uma sólida base de entendimento para a construção de um sistema de descoberta de conhecimento em banco de dados. Por outro lado, cada área apresenta, individualmente, um vasto campo de problemas e soluções.

Neste capítulo foram apresentados alguns conceitos básicos sobre as áreas relacionadas ao objetivo central deste trabalho. É possível encontrar vários trabalhos que se aprofundam tratam dos conceitos destes conceitos separadamente.

Por se tratar de uma proposta de uma arquitetura de software para sistemas de descoberta de conhecimento em banco de dados, este trabalho reuniu todos estes conceitos para o levantamento dos requisitos da arquitetura. Desta forma, buscou-se reunir em uma única arquitetura todos os elementos necessários para a construção de um sistema KDD.

Além dos conceitos relacionados ao domínio de KDD, a fundamentação teórica apresentou um item sobre as questões relacionadas ao desenvolvimento de uma arquitetura de software. Os padrões para a definição de uma arquitetura de software foram peças chaves que guiaram a realização deste trabalho de uma forma lógica e progressiva.

O próximo capítulo apresenta o desenvolvimento da arquitetura de software, mostrando cada passo de sua concepção e como os padrões foram utilizados em sua composição.

3. – A arquitetura proposta

Kimball (1998) e Menolli (2004) tratam da construção de DW. Fayyad (1996), Han e Kamber (2001) e outros autores, apresentados em Herden (2006), propõem etapas do processo KDD onde é considerado que a fonte de dados para a busca de conhecimento pode ser um DW ou bases de dados mantidas por sistemas transacionais. Estes trabalhos e ainda os trabalhos apresentados na sessão de trabalhos relacionados apresentam as etapas necessárias na busca de conhecimento e algumas ferramentas. No entanto, atualmente, nota-se uma carência de arquiteturas de software que facilitem o desenvolvimento de sistemas KDD e possibilitem a integração da construção de um DW ao processo KDD, sem burocratizar o processo KDD e, ao mesmo tempo, oferecendo opções para que cada instituição possa adaptá-las à sua realidade.

A importância de uma arquitetura em qualquer desenvolvimento de software é a mesma importância que é dada às plantas e projetos de uma construção civil. Kazman et al (2000) afirmam que a arquitetura é determinante para a qualidade de um software. Assim, o entendimento dos componentes da arquitetura por parte da equipe de desenvolvimento também é importante para que se alcance a qualidade esperada.

Seguindo a proposta para a composição de uma arquitetura de software de Bass et al (2003), o primeiro passo é a definição de um modelo de referência da aplicação. Na definição do modelo de referência para sistemas KDD são tomados como base os trabalhos de Kimball (1998), Menolli (2004) e os processos KDD apresentados em Herden (2006), descritos no capítulo anterior. Este modelo captura as principais funcionalidades dos sistemas KDD e de OLAP, além das atividades realizadas na construção de um DW, e organiza-as em partes maiores (módulos) mostrando o relacionamento entre estas partes e como elas podem ser utilizadas juntas.

Este capítulo apresenta, inicialmente, o modelo de referência proposto para sistemas KDD e considerações sobre a escolha do estilo arquitetural mais adequado para o tipo de aplicação. Na seqüência é representada a arquitetura de referência em diferentes visões e é descrita a arquitetura de software proposta neste trabalho. Para finalizar são apresentadas as considerações finais do capítulo.

3.1. Modelo de referência proposto para sistemas de KDD

O modelo de referência proposto neste trabalho possibilita a integração das funcionalidades necessárias para a construção de um DW, de sistemas OLAP e de sistemas KDD em um único sistema. Isto proporciona um alto grau de reuso dos componentes do sistema e, conseqüentemente, torna mais viável sua construção comparada com propostas que isolam a construção de um DW destes outros sistemas. Na Tabela 3.1 são relacionadas as atividades geralmente necessárias no desenvolvimento de sistemas OLAP e de sistemas KDD. Essas atividades podem ser classificadas para esses tipos de sistemas como:

- Relevante: Quando a atividade está diretamente ligada aos objetivos do sistema;
- Útil: Quando a atividade pode ser bastante útil para o sistema apesar de não ser obrigatória;
- Não aplicável: Quando a atividade não possui relação alguma com os objetivos do sistema.

Tabela 3.1: Aplicabilidade de atividades aos sistemas de OLAP e KDD.

Tipos de sistemas / Atividades	KDD	OLAP
Acesso às bases operacionais	Relevante	Relevante
Limpeza e transformação de dados	Relevante	Relevante
Definição de uma área intermediária para armazenamento de dados limpos e homogeneizados	Útil	Útil
Armazenamento de dados em estruturas multidimensionais (DW)	Útil	Relevante
Processamento analítico dos dados para geração de informação	Não aplicável	Relevante
Aplicação de técnicas para mineração de conhecimento	Relevante	Não aplicável
Armazenamento de resultados processados	Útil	Útil
Técnicas de visualização de informações	Útil	Útil

Na Tabela 3.1 é possível observar que a maioria das atividades pode ser comum aos dois tipos de sistemas. Assim, integrar conceitos de sistemas KDD e de OLAP não é tentar unir coisas muito distintas, pelo contrário, possibilita a construção de um sistema mais completo de busca de informações e conhecimentos em banco de dados. Vale destacar que do ponto de vista de uma empresa ou instituição, o desenvolvimento de um sistema que integre KDD e OLAP oferece uma importante ferramenta para a tomada de decisão.

O modelo de referência proposto é apresentado na **Figura 3.1**. Nele foram definidas três fontes diferentes de dados que podem ser utilizadas no processamento OLAP ou na

descoberta de conhecimento em banco de dados: dados operacionais, área de estágio e DW. Com isto, o modelo se apresenta bastante versátil para atender às necessidades de diversas metodologias. Isto porque, há trabalhos que propõem a construção de um DW sem a construção de uma área de estágio, alguns especificam uma área de estágio e outros ainda definem o processamento dos dados diretamente das bases operacionais. Este modelo de referência serviu como base para a definição da arquitetura de referência e da arquitetura de software. As seções a seguir descrevem cada elemento do modelo de referência.

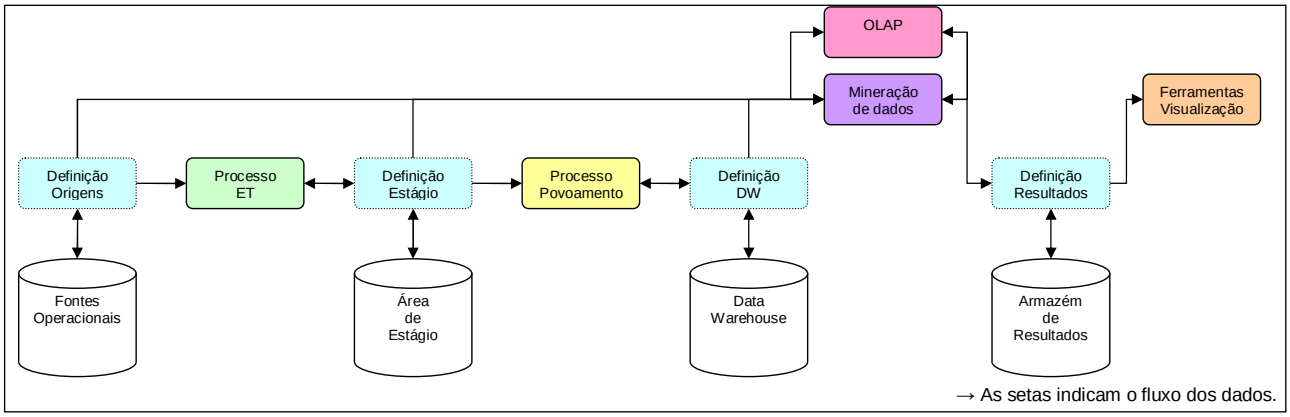


Figura 3.1: Modelo de referência para sistemas KDD.

O modelo de referência guiará a definição da arquitetura de referência e de software. As seções a seguir descrevem cada elemento do modelo de referência.

Bases de dados

Tratando-se de um sistema de descoberta de conhecimento em banco de dados, as bases de dados são os elementos estritamente necessários para a aplicação. O modelo de referência proposto por este trabalho define quatro tipos diferentes de bases de dados, relacionadas a seguir:

1. Fontes Operacionais: Essas bases são as fontes dos sistemas transacionais (também conhecidos como sistemas legados). Essas bases são acessíveis por uma conexão de dados;

2. Área de Estágio: Esta base auxilia no processo de integração dos dados e povoamento do DW, mantendo os dados extraídos das fontes operacionais que passaram por algum processo de transformação até que eles sejam carregados no DW pelo processo de povoamento. Neste modelo, a área de estágio ganha uma nova funcionalidade além de ter o objetivo de intermediar o povoamento do DW. Esta área pode ser usada como

uma fonte de origem de dados para aplicação de técnicas de mineração de dados. Isto porque os dados nela contidos foram extraídos das bases operacionais e transformados, podendo ser utilizados na busca de informações e conhecimento;

3. Data Warehouse: Esta base é um grande armazém de dados definido no modelo multidimensional. Freqüentemente, esta base é utilizada para operações de OLAP, e pode ser usada na mineração de dados;

4. Armazém de Resultados: Esta base é usada para armazenar os resultados dos processos OLAP e de mineração de dados, possibilitando futuras análises através de ferramentas de visualização e servindo como um importante repositório histórico das informações e conhecimentos gerados. Os resultados armazenados podem ser utilizados como origem para novas execuções de OLAP ou de mineração de dados. Por esta razão, há um canal bidirecional ligando o elemento de mineração de dados e de OLAP com o armazém de resultados. A vantagem desta ligação é que dados já processados por operações de OLAP podem ser utilizados em novas execuções de mineração. Isto permite usar OLAP como um recurso a mais para preparação de dados e execução de mineração, além das já definidas na extração, transformação e carga do DW.

Um dos principais diferenciais deste trabalho é a definição destas quatro fontes alternativas de dados para utilização nos processos OLAP e de mineração de dados. Isto torna o ambiente flexível para que o analista do sistema KDD, ao utilizar o modelo proposto, escolha qual repositório de dados é mais adequado para sua necessidade. Para a definição destas fontes de dados teve-se a visão de que os dados podem estar em quatro estados diferentes dentro do modelo proposto, como segue:

- Dados crus: São os dados nas bases operacionais. Estes dados podem ser utilizados diretamente como fonte dos processos OLAP se não estiverem distribuídos em diversas bases, ou ainda, se o objetivo é analisar somente uma das bases disponíveis. Já para sua utilização nos processos de KDD, acredita-se que estarão em formatos não apropriados, sendo necessário enviá-los para a área de estágio e o DW antes de se executar alguma técnica de mineração de dados sobre eles.

- Dados limpos e integrados: Neste estado, os dados foram obtidos das bases operacionais e estão limpos, transformados e integrados em uma única base (área de estágio). Esses dados já poderão estar no formato apropriado para mineração.
- Dados no formato multidimensional: Esses dados encontram-se no DW e são otimizados para os processos OLAP que exploram estruturas multidimensionais. Neste estado, os dados podem ser utilizados na execução de técnicas de mineração.
- Dados processados por OLAP ou mineração: Esses dados encontram-se no armazém de resultados e foram produzidos pelas execuções de operações de OLAP e de mineração. Esses dados podem ser úteis para a realização de novas execuções de operações de OLAP ou de mineração.

A administração das bases e dos dados nelas contidos é responsabilidade do analista que estará utilizando este modelo de referência. Busca-se aqui apresentar um modelo bastante versátil, com algumas inovações, porém, com as características básicas de qualquer processo KDD e OLAP.

3.1.1. Definição das bases de dados

Para que uma base de dados possa ser acessada é necessário que o sistema conheça sua organização, ou seja, que haja informações sobre sua localização, quantas tabelas de dados existem e quais são os nomes e os tipos das suas propriedades. Estas informações possibilitam o sistema interagir com seu usuário de forma a mostrar-lhe quais dados estão disponíveis.

Cada processo de definição, de uma forma geral, cria estruturas de dados que descrevem a base de dados pela qual é responsável. Estas estruturas são utilizadas pelos outros processos para o acesso e a manipulação das bases disponíveis. Das quatro bases identificadas na arquitetura, as definições são:

- Fontes Operacionais: são descritas quais tabelas existem nas fontes operacionais, quais as propriedades destas tabelas e os tipos das propriedades;
- Área de Estágio: são descritas as tabelas, as propriedades e os tipos das propriedades que compõem a área de estágio;
- Data Warehouse: são descritos os fatos e as dimensões que compõem o DW;

- Armazém de Resultados: São descritas as estruturas que são utilizadas para o armazenamento dos resultados OLAP e de MD;

3.1.2. Processo de ET

O processo de ET (Extração e Transformação) agrupa as atividades de busca e transformação dos dados das bases operacionais para o seu armazenamento na área de estágio. Como pode ser visto na **Figura 3.1**, os dados da área de estágio podem ser usados para o povoamento do DW pelas técnicas de mineração de dados, ou ainda, podem servir de origem para processos OLAP que utilizem dados relacionais e não multidimensionais.

A integração dos dados a serem usados tanto em processos OLAP como em KDD exige que sejam desenvolvidas funções de transformação para os dois modelos de processo. No entanto, muitas destas funções são compatíveis para os dois modelos, como por exemplo: função de tratamento de valores nulos, função de homogeneização de valores, função de formatação de dados, etc (Kimball, 1998). Já funções para geração de amostragens estatísticas de dados é mais útil para técnicas de mineração de dados. A reunião destas funções em um único módulo de processo ET centraliza e padroniza a implementação destas funções.

3.1.3. Processo de Povoamento

O objetivo deste processo é povoar o DW com os dados existentes na área de estágio. As principais funções deste processo são sumarização, agrupamento de informações e registro dos dados no DW. A definição da granularidade dos dados armazenados no DW influencia todas as atividades do povoamento. Inmon (1997) discute sobre os diversos aspectos e os impactos desta definição para o projeto do DW.

Do ponto de vista de estruturas, este processo relaciona os dados da área de estágio com as funções de povoamento e define o destino dentro do DW.

3.1.4. OLAP e Mineração de Dados

Estes dois módulos são centrais para a arquitetura proposta por este trabalho. Eles integram em um único ambiente os processos de OLAP e MD, reutilizando as operações de extração, transformação e carga e compartilhando repositórios de dados. Além disso, os dados de entrada desses módulos podem vir diretamente das bases operacionais, da área de estágio ou

do DW, o que possibilita que este modelo seja utilizado na implementação de diversos sistemas KDD.

As estruturas que definem as fontes operacionais, área de estágio e DW são utilizadas por estes módulos para que eles conheçam a organização de cada base e saiba como e o que pode ser acessado. Além destas bases, o módulo de mineração ainda pode usar como entrada de dados os resultados OLAP que se encontram no armazém de resultados. Isto possibilita a aplicação de técnicas de mineração em dados resultantes dos processos OLAP.

3.1.5. Ferramentas de visualização

A principal função deste módulo é oferecer funcionalidades que permitam que ferramentas de visualização e análise se integrem ao sistema KDD.

Atualmente, está sendo desenvolvido um trabalho de dissertação de mestrado que tem como proposta a definição de técnicas de visualização para sistemas KDD (Rabelo, 2006). Os resultados deste trabalho permitirão um refinamento das características deste módulo.

3.1.6. Requisitos não funcionais

Acima foram descritos os módulos e suas principais funções. Nesta seção são descritos os requisitos não funcionais que foram levantados durante a revisão bibliográfica e reuniões do grupo de pesquisa para a definição do modelo de referência.

Como requisitos não funcionais de um sistema computacional consideram-se principalmente questões de qualidade de software. Bass et al (2000) apresenta algumas primitivas de qualidade que devem ser consideradas no desenvolvimento de um software. Alguns anos depois, O'Brien et al (2005) destacam as vantagens da implementação de uma arquitetura orientada a serviços. Estes dois trabalhos foram utilizados como base para definir as questões de qualidade da arquitetura de referência. O segundo, em especial, contribuiu para a definição dos serviços e processos de negócio e de uma arquitetura para suportar estes componentes. A seguir são relacionadas algumas questões de qualidade esperadas para a arquitetura proposta:

- Portabilidade de plataforma: Isto significa que o sistema KDD deverá ser compatível com diversos sistemas operacionais. A principal vantagem desta compatibilidade é tornar o sistema disponível ao maior número de empresas e instituições possível, uma vez que seu uso não fica restrito a apenas um sistema operacional, proprietário ou não.

Hoje no Brasil, o governo incentiva a utilização de software livre[1] e, conseqüentemente, o sistema operacional Linux[2]. No entanto, outros sistemas operacionais proprietários ainda fazem parte da estrutura computacional da maioria das empresas e instituições.

- Portabilidade de interface: Possibilidade de implementação de diversas tecnologias de interfaces. Kimball (1998) propõe uma arquitetura com duas grandes partes: *back-room* e *front-room*. A primeira é onde se encontra a definição da estrutura de extração, transformação e povoamento do DW e a outra é onde se encontram as ferramentas de consulta e visualização. Para a segunda é sugerida a criação de um portal *on-line*, ou seja, tecnologia WEB. Este trabalho propõe que toda a lógica do negócio KDD seja definida independentemente da tecnologia de interface. Assim, tanto aplicações de *back-room* quanto de *front-room* poderão ser desenvolvidas com tecnologia WEB, ou ainda, com a tecnologia que a equipe tiver maior experiência.

- Portabilidade de dados: Acesso a dados independente de sistema gerenciador de banco de dados. Se um sistema for desenvolvido vinculado a um sistema gerenciador de banco de dados específico, provavelmente ele não será utilizado por muitas empresas, principalmente quando se trata de um sistema KDD. Isto porque, quando uma solução KDD é implantada dentro de uma instituição ela deverá extrair dados dos sistemas legados existentes para gerar algum conhecimento. Esses sistemas podem utilizar uma dentre as inúmeras tecnologias de armazenamento de dados disponíveis. Essas tecnologias vão desde projetos de código aberto até produtos que podem custar milhares de reais. Claro que desenvolver um sistema com compatibilidade universal seria um empreendimento um tanto utópico, tendo em vista que novas soluções aparecem constantemente. No entanto, existem algumas ferramentas que assumem o papel de integrador de diferentes origens de dados. Utilizando essas ferramentas, o projetista de software pode concentrar-se mais em seu domínio de negócio. Algumas dessas ferramentas são a ODBC[3] (*Open Database Connectivity*) da Microsoft, a BDE[4]

[1] www.softwarelivre.gov.br

[2] www.linux.com

[3] http://msdn.microsoft.com/library/en-us/odbc/htm/dasdkodbcoverview.asp

[4] http://info.borland.com/devsupport/dbe/

(*Borland Database Engine*) da Borland e a JDBC[1] (*Java Database Connectivity*) da Sun Microsystems.

- Segurança: Kimball (1998) destaca que desde os primeiros momentos do desenvolvimento de um sistema de DW, as questões de segurança devem ser abordadas. O mesmo ocorre para sistemas KDD que são responsáveis pela manipulação de dados, informações e conhecimentos sobre o negócio da organização. A seguir são listadas algumas questões de segurança que fazem parte da arquitetura proposta neste trabalho para sistemas KDD:
    - Cada usuário deverá ter acesso somente às funcionalidades do sistema às quais esteja autorizado: Esta restrição é importante para evitar que usuários não autorizados alterem as definições das estruturas internas do sistema KDD que a equipe definiu, como também, restringe o acesso aos resultados dos processos OLAP e KDD que contêm informações e conhecimentos privados. Esses resultados somente podem ser analisados por pessoas autorizadas.
    - A existência de uma estrutura de segurança que facilita a definição de direitos de acesso de novos componentes que sejam adicionados ao sistema. Por ser um modelo versátil não se espera definir alguma coisa fechada e restrita, ao contrário, pretende-se apresentar um modelo que possa ser extensível e adaptável. Com isto, é importante que os novos componentes que venham a ser integrados ao modelo possam também ser facilmente integrados às estruturas de segurança.
    - Registro das operações: Este requisito é importante principalmente para a aplicação deste modelo em um ambiente com um grande número de pessoas tendo acesso ao sistema KDD. O registro das operações realizadas pelo sistema é uma importante ferramenta para que gerentes e analistas acompanhem como o sistema está sendo utilizado, identificando os módulos com mais acesso, ou ainda, quem está usando o quê e onde.

[1] http://java.sun.com/javaset/technologies/database.jsp

- Manutenibilidade: o sistema poderá receber novas funcionalidades, ou ainda ter as atuais modificadas de acordo com a necessidade da instituição e da equipe de desenvolvimento.
- Desempenho e Usabilidade: o sistema deve atender às requisições dos usuários em um tempo que não cause impaciência nos mesmos. Ou ainda, durante a execução de atividades prolongadas, o sistema deve informar a atual situação de execução da atividade para que o usuário possa acompanhar o andamento. Qualquer problema deverá ser informado ao usuário descrevendo, de uma maneira clara, o que há de errado e como ele pode proceder para corrigir o problema. O sistema deve ser de fácil aprendizagem, sendo que, para isto devem ser utilizados termos comuns ao domínio KDD em sua interface, além de uma apresentação lógica das fases do processo e das estruturas que devem ser definidas em cada fase.
- Confiabilidade: Vários usuários poderão utilizar o sistema ao mesmo tempo, com isto, as atividades de um usuário não devem interferir na seção de outros usuários. Se o usuário executar alguma atividade e parar de responder, o sistema deve permitir que a seção deste usuário seja finalizada e uma nova seção possa ser iniciada. As atividades devem ser executadas em um ambiente transacional que garanta a consistência das informações persistidas. Caso um processo de extração, transformação ou carga seja interrompido, o sistema deverá oferecer algum mecanismo que permita reiniciar o processo a partir do ponto de interrupção.
- Disponibilidade: O sistema KDD não é um sistema crítico, ou seja, os processos de negócio não rodam dentro dele. Ele é uma ferramenta para busca de informações e conhecimentos implícitos. As fontes de dados do sistema KDD possuem dados históricos, o que implica em um grande volume de dados. Um mecanismo de redundância de dados ou backup pode ser utilizado para evitar perdas de dados por falha de hardware.

Além destes requisitos de qualidade, verificou-se a necessidade do registro das definições realizadas pelo usuário durante os processos de preparação de dados, para evitar que este tipo de trabalho seja repetido sempre que o usuário for realizar uma busca de informações ou conhecimentos usando o sistema KDD. Os módulos de definição e de processo deverão armazenar metadados para manter as configurações definidas pelo usuário e permitir sua reutilização sempre que for necessário. Os metadados definem todas as estruturas utilizadas

no sistema KDD. Podem ser desenvolvidos processos que utilizam esses metadados para automatizar a extração, transformação, carga, aplicação de técnicas OLAP e MD e armazenamento dos resultados, diminuindo a interação do usuário, principalmente quando é grande o volume de dados a ser processado.

3.2. A escolha do estilo arquitetural

Em um primeiro momento, o estilo arquitetural *pipes and filters* foi considerado um provável candidato a ser escolhido. Esta impressão se deve ao fato de que alguns processos KDD, apresentados em (Herden, 2006), propõem uma seqüência de passos bem definidos que, em geral, inicia com a fase de pré-processamento onde é realizada a preparação de dados (limpeza, extração e transformação) , segue com a mineração de dados, e, por fim, a fase de pós-processamento que dá suporte à análise dos resultados obtidos. Estes passos poderiam ser definidos como filtros dos dados

No entanto, analisando a execução de um sistema KDD, observa-se que durante todo seu ciclo de vida a interação do usuário é essencial. O usuário necessita: analisar as bases de origem e determinar o que pode ser útil para o seu processamento; organizar os dados em áreas de estágio ou ainda em um modelo multidimensional; escolher as técnicas de MD ou OLAP que serão aplicadas aos dados; preparar o conjunto de dados que será processado; definir como armazenar e visualizar os resultados obtidos. Em outras palavras, o processo KDD é indeterminístico (Dias, 2001). Muitas tentativas são realizadas antes de se obter um modelo de dados adequado. Ou seja, De acordo com Shaw e Garlan (1996), o estilo arquitetural *pipe and filters* não são indicados para aplicações interativas.

No processo KDD não existe uma rigidez na seqüência em que os passos devem ser realizados. Um sistema KDD torna-se bastante útil quando se apresenta como um conjunto de ferramentas e serviços que interagem com o usuário e o auxilia na sua execução. É importante que alguns problemas de tecnologias e estruturas sejam abstraídos.

Dentre os estilos arquiteturais apresentados por Shaw e Garlan (1996), o estilo em camada oferece níveis de abstração que permitem dividir um problema complexo em uma seqüência de passos incrementais e suporta reuso. Estas características tornam este tipo de estilo adequado a sistemas complexos e interativos como é o caso do sistema KDD (Dias, 2001). A característica de reuso deste estilo de arquitetura é um aspecto importante que foi levado em

consideração neste trabalho por ter sido necessário definir uma arquitetura de referência para se chegar à definição da arquitetura de software proposta para sistemas KDD.

No modelo de referência proposto pode-se observar que algumas funcionalidades são compartilhadas entre os módulos do modelo. Como por exemplo, as rotinas de acesso às bases de dados. Essas funcionalidades poderão ficar em uma camada mais inferior que seja acessível pelas camadas superiores.

Analisando os processos definidos no modelo de referência, nota-se que são necessários alguns níveis de abstração que resolvam alguns problemas de baixo nível e forneçam soluções mais simplificadas para os processos. Como por exemplo, o acesso a repositórios de dados possui problemáticas de conexão com servidores de banco de dados com diferentes tecnologias de armazenamento. Porém, os componentes dos níveis mais altos da aplicação estão interessados em simplesmente acessar os dados, independentemente da tecnologia de armazenamento. Para isto, são necessários componentes nos níveis mais baixos da aplicação que realizem esta abstração e forneçam funcionalidades mais específicas para as camadas superiores.

Portanto, teve-se mais uma visão vertical, em camadas, do que uma visão horizontal, em canos e filtros, para as soluções dos problemas de organização e implementação dos componentes do sistema KDD.

3.2.1. As camadas definidas para a arquitetura

Uma vez escolhido o estilo arquitetural, agora vem outra pergunta: quantas camadas serão necessárias para a arquitetura? A resposta foi obtida definindo-se níveis de responsabilidades na aplicação.

O primeiro nível é o mais óbvio, a aplicação necessita de uma camada que controle a interação do usuário com o sistema. Logo, é definida uma camada superior de visualização. A definição dos demais níveis de responsabilidade foi influenciada pelos conceitos de SOA. Desta forma, foi definida uma camada para abrigar os processos de negócio e outra camada para os serviços da aplicação. Por último, existem algumas funcionalidades que dão suporte aos serviços e aos processos e não se enquadram nas camadas de visualização, de processo e de serviço. Para essas funcionalidades foi definida uma camada inferior, denominada camada de apoio.

A Figura 3.2 representa as camadas definidas na arquitetura. Para cada camada existem gerenciadores responsáveis por controlar a instanciação e o acesso aos objetos da camada. Alguns gerenciadores são interligados. Dentro de cada camada, os objetos têm acesso ao seu gerenciador e podem utilizá-lo para referenciar outros gerenciadores, podendo então ter acesso aos objetos controlados por esses gerenciadores. Com isto, os objetos de cada camada ficam isolados e o único meio de acesso a eles é por meio do gerenciador responsável pela camada. Esta organização oferece um baixo acoplamento entre os processos, serviços e outros componentes e, também, um maior controle de acesso às camadas, permitindo a implementação de rotinas que monitoram o comportamento de toda a arquitetura e facilitando a análise de desempenho e a busca por gargalos ou pontos críticos. Outra vantagem observada nesta proposta é que a função de cada camada fica bem explícita, além de fornecer uma interface bem definida para cada camada, facilitando o entendimento e a utilização da arquitetura proposta.

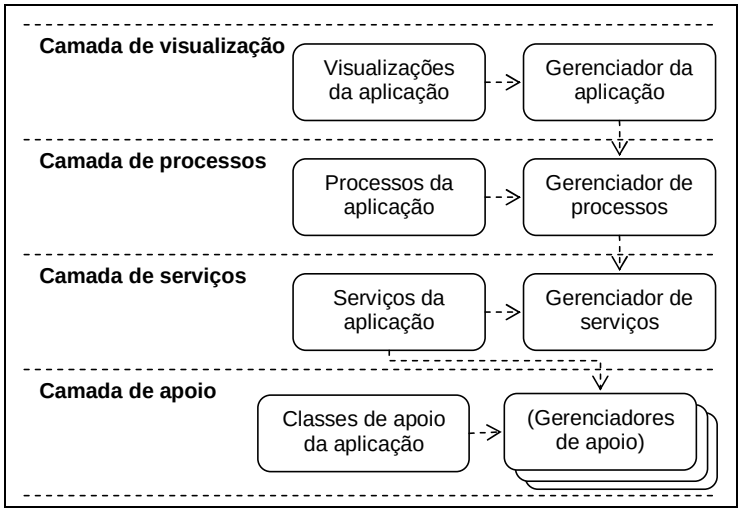


Figura 3.2: Camadas que compõem a arquitetura proposta.

Vale ressaltar que pela definição do estilo arquitetural em camadas, a camada inferior fornece funcionalidades para a camada superior e não possui qualquer referência a componentes da camada superior. No entanto, esta separação pode tornar complexa a implementação dos componentes da camada inferior, que muitas vezes poderiam utilizar uma simples referência à camada superior para facilitar a comunicação entre os mesmos. Contudo, a manutenção dos componentes de cada camada é facilitada devido a esta alta coesão e baixo acoplamento entre as camadas.

A **camada de visualização** é responsável por abrigar os objetos que exibem as interfaces (telas) e controlam a interação do usuário com o sistema. Nesta camada é aplicado o padrão

MVC. Em outras palavras, esta camada utiliza o MVC para compor suas subcamadas internas e organizar seus componentes. Esta camada é composta de dois tipos de componentes, o gerenciador de aplicação e visualizações da aplicação. O gerenciador da aplicação é responsável por fornecer à camada de visualização acesso às camadas inferiores. O outro tipo de componente representa ferramentas/técnicas de visualização a serem disponibilizadas. Um requisito é mapeado para esta camada quando o mesmo trata somente de questões de interação do usuário com o sistema.

A **camada de processos** possui um gerenciador que conhece todos os processos que são instanciados pela arquitetura. Desta forma, para acessar qualquer processo é necessário que a camada superior solicite a este gerenciador a instanciação de um determinado processo. Duas características desta camada precisam ser destacadas: 1) os processos por ela instanciados são autenticados e somente disponibilizados aos usuários que possuem o direito de acesso; 2) o processo possui uma instância específica para cada usuário que o invocou. Um requisito é mapeado para esta camada quando está relacionado a algum processo de negócio que possui início, meio e fim bem definidos. O início e o fim destes processos geralmente executam operações de controle de acesso e segurança, já as atividades intermediárias executam diversas operações que interagem com os componentes da camada de visualização e também com os serviços disponíveis na camada abaixo. Em muitos casos, as atividades intermediárias são executadas em uma ordem definida e uma atividade pode depender dos resultados obtidos nas atividades anteriores.

A **camada de serviços** também possui um gerenciador que conhece todos os serviços que são instanciados pela arquitetura e disponibiliza uma interface que permite à camada superior executar um determinado serviço. Esta camada não é autenticada e a arquitetura mantém somente uma instância de cada serviço disponível. São mapeados para esta camada os requisitos que possuem uma função bem específica que é, geralmente, utilizada em diversos processos do negócio. Alguns serviços mais complexos podem necessitar de gerenciadores de apoio para realizar suas atividades. Estes gerenciadores estão localizados na camada inferior.

A **camada de apoio** foi definida para organizar as funcionalidades do sistema que dão suporte às camadas superiores. Alguns serviços mapeados na camada superior necessitam de um gerenciamento mais específico de determinadas tecnologias para desempenhar suas funções, neste caso, um gerenciador pode ser implementado nesta camada de apoio para dar suporte à manipulação dessas tecnologias.

A visão aqui apresentada é uma visão lógica das camadas da arquitetura. Cada pacote (ou módulo), acoplado à arquitetura, implementa objetos que se integram dinamicamente em uma das camadas da arquitetura. Em outras palavras, cada pacote possui controladores de visão, processos e serviços que são instanciados durante a execução da aplicação e, quando identificados pela arquitetura, são registrados em seus gerenciadores para se tornarem visíveis para toda a aplicação. Esta integração dinâmica em tempo de execução será mais bem discutida nas próximas seções que continuam com a documentação da arquitetura proposta.

3.2.2. Exemplo ilustrativo

Para ilustrar é apresentado como um determinado caso de uso é realizado através da implementação das suas atividades em diferentes camadas.

A definição das origens dos dados é um caso de uso do sistema KDD proposto e está representado no modelo de referência. Seu objetivo é permitir que o usuário defina quais tabelas e itens de dados das bases operacionais estarão disponíveis para as operações de extração, carga, mineração ou OLAP. Suas atividades principais são:

1. Listar as conexões com gerenciadores de banco de dados disponíveis para o usuário;
2. Permitir ao usuário selecionar uma conexão;
3. Listar as tabelas da conexão selecionada pelo usuário;
4. Permitir ao usuário selecionar uma tabela;
5. Listar os itens de dado da tabela selecionada pelo usuário;
6. Permitir ao usuário selecionar vários campos;
7. Definir os itens de dado selecionados como disponíveis para as outras atividades do sistema KDD;

Tomando esta descrição simplificada do caso de uso, observam-se três características importantes que auxiliam na identificação dos tipos de requisitos que ele gerará para a arquitetura. A primeira está relacionada à existência de pontos de partida e de chegada bem definidos, no caso, a atividade 1 inicia o caso de uso e a atividade 7 encerra. A segunda é o fato de uma atividade depender dos resultados das atividades anteriores como por exemplo, a

atividade 2 usa os resultados da atividade 1. E a terceira é a presença constante de uma interação com o usuário, ou seja, deve haver uma interface que permita ao usuário visualizar as listas e selecionar itens (atividades 2, 4 e 6). A primeira e a segunda características estão relacionadas à existência de um processo que possui um início de execução, atividades intermediárias interativas e um final de execução. A terceira característica está relacionada à necessidade de componentes de interação para a exibição e seleção de dados. Tem-se até aqui a separação das funcionalidades da camada de visualização e da camada de processos.

Analisando melhor cada atividade, é possível observar que são atividades bem específicas. Elas geram algumas informações que auxiliam no andamento do processo de negócio. Por exemplo, uma atividade que lista as conexões com os gerenciadores de banco de dados disponíveis, com certeza não será utilizada somente por este processo de negócio. Logo, não deve ser implementada dentro deste processo, mas em um nível inferior que possa ser reutilizada. Esta atividade deverá ser definida como um serviço de negócio que será localizado na camada de serviços.

Um serviço de negócio que lista as conexões com os bancos de dados precisará consultar alguma estrutura que mantém as configurações de conexões disponíveis no sistema. Logo, este serviço precisará acessar um mecanismo responsável pela persistência de objetos. Seguindo as camadas definidas, este mecanismo deve ser incluído na camada de apoio. Estando nesta camada, outros serviços também poderão acessar este mecanismo de persistência para obter instâncias de outros objetos desejados, aumentando, consideravelmente, seu grau de reuso.

Portanto, cada caso de uso define um ou mais processos e cada atividade sua define os componentes de visualização, serviço e apoio que serão necessários para sua realização. Na Tabela 3.2 estão relacionadas as atividades e os componentes necessários em cada camada.

Tabela 3.2: Relação de atividades do caso de uso com a definição dos elementos de cada camada.

Atividade	Camada de visualização	Camada de serviço	Camada de apoio
1. Listar as conexões com gerenciadores de banco de dados disponíveis para o usuário	Componente visual para exibição de lista	Serviço que obtenha as conexões disponíveis	Mecanismo de persistência de objetos para recuperação futura
2. Permitir ao usuário selecionar uma conexão	Componente visual para seleção de itens de uma lista		

3. Listar as tabelas da conexão selecionada pelo usuário	Componente visual para exibição de lista	Serviço que obtenha as tabelas de uma determinada conexão	Componente para acesso ao banco de dados definido pela conexão
4. Permitir ao usuário selecionar uma tabela	Componente visual para seleção de itens de uma lista		
5. Listar os itens de dado da tabela selecionada pelo usuário	Componente visual para exibição de lista	Serviço que obtenha os itens de dado de uma tabela	Componente para acesso ao banco de dados definido pela conexão
6. Permitir ao usuário selecionar vários campos	Componente visual para seleção de vários itens de uma lista		
7. Definir os itens de dado selecionados como disponíveis para as outras atividades do sistema KDD		Serviço que persista a seleção de propriedades de uma tabela	Mecanismo de persistência de objetos para recuperação futura

3.3. Arquitetura de referência

A definição de uma arquitetura apresenta uma série de conceitos e estruturas que tentam melhorar o entendimento do sistema e organizar o seu desenvolvimento. No entanto, a maioria dos aspectos arquiteturais são restrições conceituais, ou seja, fisicamente não restringem o desenvolvedor de infringi-las. Assim, se o desenvolvedor não tiver um conhecimento razoável do que é proposto pela arquitetura, o sistema desenvolvido poderá ter seus aspectos de qualidade comprometidos.

Seguindo o estilo arquitetural e as camadas definidas para este trabalho, os requisitos do sistema KDD foram divididos em requisitos de interface, de processo, de serviços e de apoio. O problema é definir o que exatamente deve ser designado para cada camada; isto não é uma tarefa muito fácil, principalmente para pessoas que estão iniciando no uso da arquitetura. Durante a implementação do protótipo deste trabalho pôde-se observar que muitas vezes pode ocorrer do desenvolvedor implementar rotinas como pertencentes a determinados componentes da arquitetura que, na verdade, pela definição, deveriam estar em outros componentes. Para que isto não aconteça é muito importante que as estruturas e as responsabilidades estejam sempre bem definidas na arquitetura, e quando surgir a pergunta onde eu devo implementar isto ou aquilo, a documentação da arquitetura deverá ser consultada.

Diante da problemática exposta acima, nesta seção é descrita a arquitetura de referência que foi definida para o sistema KDD e os princípios que guiaram sua concepção. É importante

ressaltar que a arquitetura de referência é um passo anterior à definição da arquitetura de software.

A arquitetura de referência aqui apresentada busca atender os requisitos de um sistema KDD, mas seu uso não é limitado a este tipo de sistema. Isto é, os problemas por ela abordados e as soluções por ela definidas podem ser encontrados em outros domínios de aplicação. Neste caso, a arquitetura de referência poderá ser completamente ou parcialmente reutilizada. Esta abordagem proporciona o reuso de arquiteturas de referência entre diversos projetos, lembrando que reuso é um ponto chave, atualmente, no desenvolvimento de qualquer solução em software.

Para a documentação da arquitetura de referência foi realizada uma adaptação da metodologia apresentada por Merson (2006). Esta metodologia define que uma arquitetura pode ser documentada em termos de visões. Cada visão destaca aspectos estáticos ou dinâmicos da arquitetura. Nem todas as visões são necessárias para todos os projetos.

As seções que seguem apresentam como a arquitetura de referência foi concebida e sua documentação.

3.3.1. Concepção

A concepção da arquitetura de referência é influenciada diretamente pelo estilo arquitetural escolhido. A organização básica dos componentes da arquitetura e a responsabilidade de cada camada são pré-estabelecidos pelo estilo arquitetural em camadas e, também, já foram descritos nas seções anteriores. No entanto, algumas estruturas de controle precisam ser definidas para especificar como serão realizadas as comunicações entre as camadas e o gerenciamento das estruturas internas de cada camada. Além disso, os requisitos não funcionais ainda precisam ser analisados para verificar o seu impacto na arquitetura proposta, principalmente os requisitos de qualidade esperados.

Somente o fato de definir uma arquitetura para um software já contribui para a avaliação positiva de suas questões de qualidade. Isto porque a arquitetura rege todo o desenvolvimento, integração e evolução do software. A seguir são relacionados os requisitos não funcionais apresentados na seção do modelo de referência e a influência de cada requisito sobre a concepção da arquitetura de referência.

Armazenar definições realizadas pelo usuário

Este requisito é um diferencial desta proposta. Ele define que os processos executados no sistema devem armazenar estruturas e configurações para evitar que seja necessário realizar a preparação de dados e demais definições toda vez que o usuário desejar buscar por informações ou conhecimentos nas bases de dados disponíveis. Com isto, foram definidas estruturas que descrevem as bases de dados de origem, a área de estágio, o DW, o armazém de resultados, as operações de extração, transformação e povoamento, as configurações das técnicas de mineração e operações de OLAP. Estas estruturas são persistidas e um mecanismo avançado de abstração de **entidades**[1] é responsável por fazer a manutenção destas estruturas. Este mecanismo foi chamado de módulo CRUD, que é a sigla do nome em inglês para as operações de criar (*create*), recuperar (*retrieve*), atualizar (*update*) e excluir (*delete*) uma entidade. O principal objetivo do mecanismo CRUD é unir informações de controle (metadados) às instâncias de objetos e a principal vantagem desta união é a implementação de máquinas que manipulem as entidades utilizando essas informações. Com isto, algumas informações sobre validação, relacionamento e restrição podem ser inseridas em um único local. Outra característica deste mecanismo é que ele mantém informações sobre os relacionamentos entre as entidades. Isto permite o acesso às entidades através de seus relacionamentos. Aplicando esta característica ao domínio KDD, é possível, por exemplo, identificar as origens das informações armazenadas no DW e verificar de quais estruturas do estágio elas vieram. Maiores detalhes sobre este mecanismo e suas vantagens são apresentados no item 4.3.3.

O mecanismo CRUD faz parte da arquitetura de referência por ser uma solução não restrita ao domínio de sistemas KDD, já as estruturas que descrevem as bases e as operações são específicas ao domínio tratado e são descritos na seção da arquitetura de software.

[1] Entidades: este termo possui o mesmo significado aplicado às entidades do modelo relacional. Uma entidade é um objeto que existe e é distinguível de outros objetos. Uma entidade pode ser um objeto com uma existência física (entidade concreta) – um empregado, pessoa, carro, casa em particular – ou conceitual (entidade abstrata) – uma companhia, um emprego, um curso universitário. Neste trabalho, como é utilizada a abordagem orientada a objetos para modelagem da arquitetura, uma entidade se refere a uma classe com atributos privados e com métodos públicos para definição e leitura destes atributos.

Portabilidade de plataforma

A portabilidade de plataforma está diretamente ligada à tecnologia de implementação e às ferramentas a serem utilizadas na implementação do sistema. No entanto, a especificação de um sistema pode apresentar requisitos funcionais que exijam a utilização de uma determinada tecnologia que pode estar disponível somente em uma linguagem específica. Com isto, este requisito não funcional pode não ser satisfeito. Neste trabalho, este requisito influenciou a escolha da tecnologia Java[1] para a implementação do protótipo. No entanto, a proposta da arquitetura de referência é independente da linguagem de programação escolhida, ou seja, não foi encontrado algum aspecto que impeça a implementação da arquitetura em outra linguagem.

Portabilidade de interface

Este requisito pode ser completamente atendido se o projetista da arquitetura tiver em mente a proposta do padrão MVC. A principal exigência deste requisito é que todas as rotinas sejam implementadas de forma independente da tecnologia de interface. É comum em sistemas sem arquitetura que muitas regras de negócio sejam implementadas em eventos ou classes de interface. Desta forma, as regras ficam praticamente sem chance de ser reusadas e qualquer alteração na interface pode corromper ou alterar completamente a lógica do negócio.

Este requisito influenciou a definição de uma camada de visualização e uma camada de processos que é totalmente independente da tecnologia de visualização. A camada de visualização é responsável pelo controle MVC. Estas duas camadas são representadas na Figura 3.3 como sendo os módulos *view* e *process*, respectivamente.

Portabilidade de dados

Tratando-se de um sistema KDD, há dois tipos de acesso a dados. O primeiro é o acesso aos dados que serão extraídos, transformados e analisados. Esses dados geralmente se encontram em gerenciadores de banco de dados robustos. O segundo tipo de acesso é o mecanismo que armazena as estruturas e configurações do próprio sistema. Este mecanismo, que foi chamado

[1] java.sun.com

CRUD, é responsável pela persistência dos objetos de dados utilizados pelo sistema. Para esta persistência é utilizado um gerenciador de banco de dados.

O acesso que se refere às origens dos dados para análise é bem específico ao domínio de sistemas KDD. Com isto, este requisito deverá ser satisfeito na definição dos componentes da arquitetura de software, que vem depois da definição da arquitetura de referência.

Para que este requisito fosse satisfeito no segundo tipo de acesso a dados, foi definido um mecanismo CRUD, já apresentado anteriormente, que centraliza as operações de criação, obtenção, alteração e remoção, permitindo assim que o problema no tratamento de diversas tecnologias de acesso a dados fosse resolvido por um componente central. A centralização da persistência de objetos facilita a implementação deste requisito por deixar a cargo de um único componente o controle da troca de tecnologia de acesso aos dados.

Segurança: Cada usuário deverá ter acesso somente às funcionalidades do sistema às quais esteja autorizado

Este requisito apresentou um dilema: O que na arquitetura deveria ser protegido por autenticação: serviços, processos ou interfaces? Primeiramente foi cogitada a definição de serviços autenticados. No entanto, logo nos primeiros momentos percebeu-se que esta definição poderia apresentar alguns problemas. Durante a execução, um processo de negócio poderia ficar impossibilitado de concluir suas atividades caso o usuário não tivesse direito de acesso a todos os serviços referenciados. Seria constrangedor para o sistema informar o usuário, no meio do processo, que ele teve acesso para iniciar, mas que não será possível concluí-lo por falta de direito de acesso a um determinado serviço. Outro problema desta definição é que o cadastramento dos direitos de acesso ficaria mais complexo e exigiria um conhecimento mais profundo das estruturas internas do sistema na hora de se definir os direitos de acesso para um usuário. Além disso, serviços são implementados para dar apoio aos processos de negócio e podem surgir de diversas necessidades não relacionadas diretamente ao negócio, mas de alguma tecnologia que está sendo utilizada pelo processo, isto dificultaria o entendimento do relacionamento do negócio com os serviços e quais usuários deveriam ter acesso a quais serviços. Logo, esta alternativa foi descartada.

Ao analisar a possibilidade dos processos serem autenticados, percebeu-se que os mesmos são mapeados diretamente dos casos de uso do sistema, isto significa que há uma relação mais estreita entre autorizar ou não um usuário a executar determinado processo. O próprio

diagrama de casos de uso do sistema identifica os perfis dos usuários (atores) que se relacionam com os casos de uso. Desta forma, foi definido que os processos da arquitetura seriam autenticados. Isto determinou que o gerenciador de processos que controla a camada de processos utilize um módulo de segurança para verificar se o usuário que está solicitando o processo possui ou não direitos de execução. Este módulo de segurança é identificado como *security* na Figura 3.3. Neste módulo foi definida uma estrutura para controlar os direitos de acesso de cada processo registrado na arquitetura.

A camada de interface também foi analisada para verificar se ela poderia ser autenticada ou não. A variabilidade das tecnologias de construção de interface foi o aspecto que mais pesou para uma negativa. Seria complicado definir uma estrutura que padronizasse a estrutura de controle de acesso considerando os diversos aspectos das tecnologias de interface. Outro aspecto importante é o fato dos processos de negócio guiarem a construção de interfaces, tornando mais favorável à definição de processos autenticados.

Concluída a definição de autenticação dos processos, apareceu outro dilema: as operações de criação, obtenção, alteração e remoção de uma entidade do sistema são realizadas por processos distintos. No entanto, o usuário pode ter direito de acesso a uma entidade A e não poder acessar uma entidade B. Pela definição anterior, tendo o usuário direito de acesso ao processo de criação, ele poderá criar qualquer entidade do sistema. Logo, como definir os direitos especificamente para cada entidade? Este dilema foi resolvido com a criação de uma estrutura para controlar individualmente os direitos de cada operação CRUD sobre cada entidade do sistema. Desta forma, a estrutura é consultada durante a criação, visualização, atualização ou exclusão de uma entidade e, caso o usuário não tenha direito suficiente, ele não conseguirá realizar a operação e será informado da insuficiência dos seus direitos de acesso.

Todas as estruturas de segurança aqui citadas são descritas com maiores detalhes no item 4.3.3 na página 113.

Segurança: Estrutura que facilite a integração de novos componentes

Este requisito garante que quando um novo componente é integrado à arquitetura, a mesma possa adicioná-lo ao controle de segurança e disponibilizá-lo ao usuário para a definição dos seus direitos de acesso. Como foi definido no item anterior, existem estruturas para o controle dos direitos de acesso dos usuários sobre processos e entidades. Estas estruturas mantêm os processos e as entidades atualmente registradas na arquitetura. Desta forma, quando um novo

processo ou uma nova entidade for integrado(a) à arquitetura, este(a) será incluído(a) em sua estrutura de segurança sem a necessidade do usuário alterar qualquer arquivo de configuração de segurança da aplicação, satisfazendo assim, este requisito.

Segurança: Registro das operações

Para atender este requisito foi proposto um módulo de auditoria (Figura 3.3) que registra todas as operações executadas pelos usuários no sistema. Porém, havia uma questão de qual operação registrar. A solução proposta seguiu o mesmo rumo das definições de autenticação de processo e segurança das entidades já discutidas. Com isto, foi definido que informações sobre as execuções dos processos e seus parâmetros de execução devem ser registrados na auditoria. Assim, a cada execução de um processo este módulo é utilizado para gerar um registro que armazena: o dia, a hora, a estação, o nome do processo, o usuário que executou o processo e alguns parâmetros adicionais que indiquem o que o usuário realizou.

Para as operações CRUD executadas sobre as entidades do sistema foi definido um registro na auditoria que armazena: o dia, a hora, a estação, o nome da entidade e a operação realizada na entidade (criação, alteração ou remoção). Caso seja executada uma operação de alteração, as propriedades alteradas são registradas na auditoria para possibilitar a identificação do que foi alterado pelo usuário. Caso seja executada a operação de remoção, uma justificativa é solicitada ao usuário e armazenada na auditoria. Com estes recursos é possível identificar o que foi alterado e por quem, ou ainda, o que foi excluído, por quem foi excluído e o motivo da exclusão. Evitando situações onde ninguém sabe quem fez o quê no sistema.

Manutenibilidade

A utilização de camadas, a separação das funcionalidades de interface e de negócio e a abstração do negócio em processos e serviços visam oferecer uma arquitetura manutenível. Uma vez que todos os componentes que estão ou serão integrados à arquitetura seguem estas abstrações, evita-se o desenvolvimento *ad-hoc* que dificulta a compreensão e a manutenção de um software.

Desempenho e Usabilidade

A utilização de camadas influencia negativamente no desempenho da arquitetura. Contudo, os processos mais críticos de um sistema KDD podem ser implementados em serviços, que por

sua vez são atômicos e independentes. Com isto, as camadas e as abstrações não produzem um impacto significativo sobre esses serviços.

Para a questão de usabilidade, foi definida uma estrutura de mensagens que permite aos componentes da arquitetura gerar informações de atenção, erro ou informação que serão exibidas ao usuário ou registradas em *log* da aplicação. Outra questão de usabilidade é a definição de processos de negócio que podem ser executados passo a passo com auxílio de um assistente. Os assistentes são bastante práticos à medida que eles guiam o usuário na definição dos parâmetros de execução de um determinado processo, fornecendo informações de auxílio sobre cada passo. Porém, para desenvolvedores que preferem compor uma interface mais carregada de controles visuais e, conseqüentemente, mais ágil, a arquitetura não restringe a execução de processos em modo passo a passo. Todos os parâmetros exigidos pelo processo podem ser preenchidos em uma única tela.

Confiabilidade

Para garantir a consistência das informações persistidas, a camada de serviço foi definida como sendo uma camada transacional, ou seja, os serviços são executados dentro de uma transação que aguarda a finalização dos serviços antes de decidir por uma operação para confirmar (*commit*) ou desfazer (*rollback*) as alterações. Como a chamada de serviços é controlada pelo gerenciador de serviços, ele é o responsável por gerenciar, também, as transações.

Disponibilidade

O sistema KDD não é um sistema crítico, ou seja, os processos de negócio não são executados por ele. Apesar da sua importância para a organização na tomada de decisão, sua indisponibilidade por algumas horas, ou até mesmo dias, não implica em paralisação do setor de produção da organização, pois estes setores utilizam outros sistemas (os transacionais). Logo, este requisito não foi trabalhado profundamente nesta arquitetura.

3.3.2. Documentação

Nesta seção é apresentada uma documentação da arquitetura de referência proposta utilizando visões de módulos, de execução, de implementação, de instalação e de comportamento. A nomenclatura dos componentes da arquitetura foi definida na língua inglesa por questão de

preferência do autor e para uma melhor correspondência entre os nomes no projeto e na implementação.

A primeira visão da arquitetura de referência é mostrada na Figura 3.3. Nesta visão são definidos os módulos e seus relacionamentos. É importante destacar nesta figura que o módulo *view* pode ser estendido por outros módulos que implementam diversas tecnologias de visualização, tornando os processos de negócio completamente independentes da tecnologia de interface.

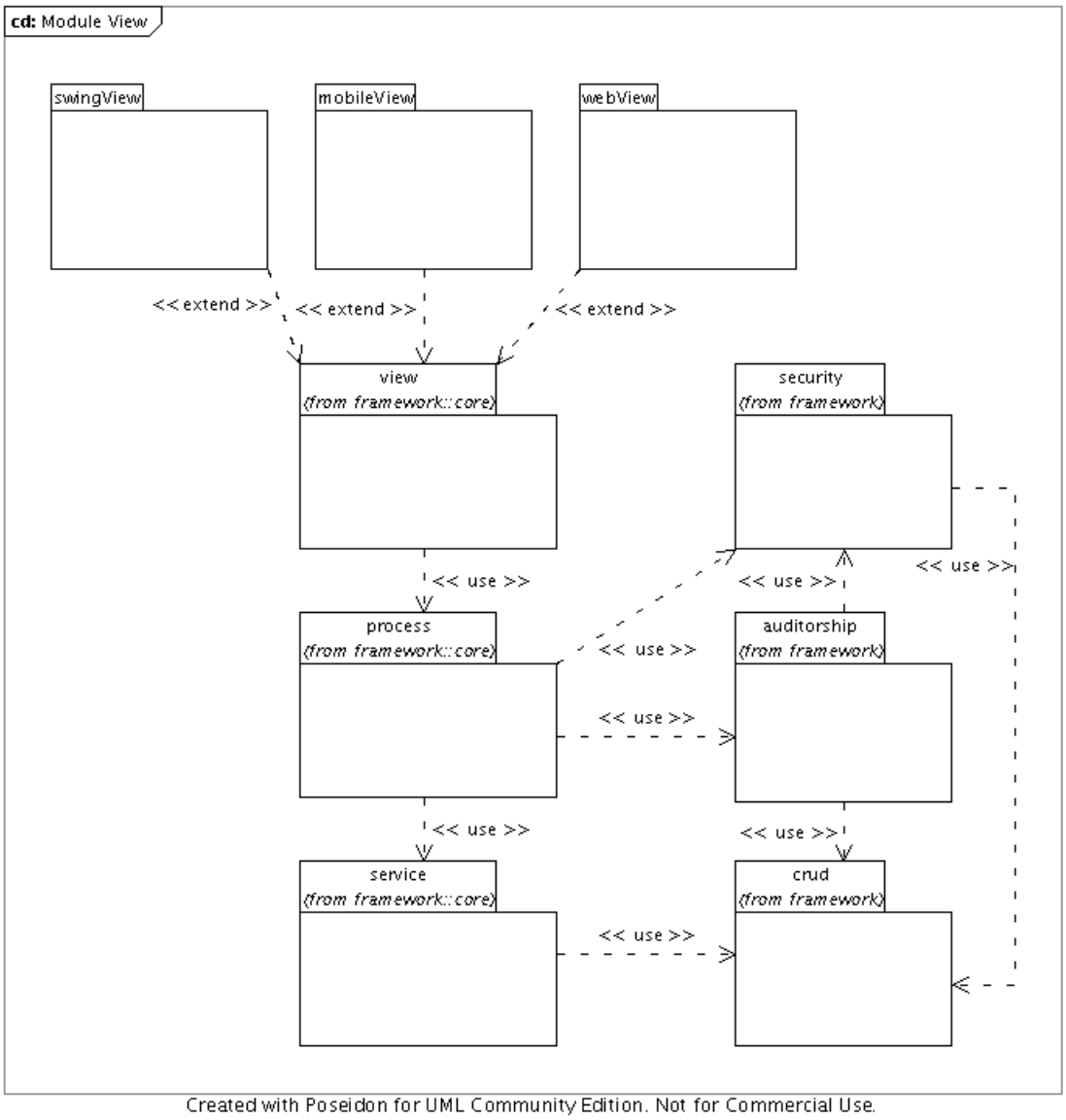


Figura 3.3: Visão dos módulos que compõem a arquitetura de referência.

A Tabela 3.3 descreve cada módulo definido na arquitetura de referência. As estruturas internas e o funcionamento dos módulos são apresentados com maiores detalhes no capítulo 4 que apresenta o framework implementado.

Tabela 3.3: Descrição dos módulos da arquitetura de referência.

Módulo	Descrição
view	Responsável por prover meios para que os controladores de interface tenham acesso aos processos de negócio e às interfaces na aplicação. Particularidades de tecnologias de interface podem ser tratadas neste módulo.
process	Este módulo é autenticado, por isto usa os módulos *secutiy* e *auditorhip* para realizar as operações de autenticação antes da instanciação de um novo processo e para registrar na auditoria informações sobre o processo executado. O gerenciador de processo que gerencia a camada de processos da aplicação encontra-se neste módulo. Há uma dependência do módulo *service*, pois os processos usam os serviços do negócio durante suas atividades.
service	O gerenciador da camada de serviços é alocado neste módulo. Neste módulo os serviços são executados dentro de um ambiente transacional. Há uma dependência do módulo *crud* para os serviços que utilizam ou manipulam entidades do sistema.
security	As entidades de segurança como usuários, grupos e direitos são incluídas neste módulo. Há também, serviços e processos para autenticação de usuário, verificação e definição de direitos de um usuário. Por manipular entidades, este módulo usa o módulo *crud*.
auditorship	Este módulo disponibiliza serviços para registrar as operações realizadas em processos e entidades.
crud	O módulo *crud* é composto de um gerenciador de entidades, serviços e processos para persistência de objetos. Porém, a principal função do módulo *crud é* fornecer uma abstração de entidade de negócio. Esta abstração consiste na união de instâncias de objetos com estruturas de metadados que contém informações sobre a entidade. Desta forma, quando uma entidade é recuperada do repositório de objetos, ela vem com os dados básicos e, ainda, mais um conjunto de informações adicionais que descrevem a entidade.

A Figura 3.4 apresenta a visão de execução (*runtime view*). Esta visão é útil para entender o funcionamento do sistema e analisar propriedades que se manifestam em tempo de execução, como desempenho e gargalos (Merson, 2006). Enquanto a visão dos módulos acima mostra os módulos separados, na visão de execução os serviços e processos podem ser vistos em suas respectivas camadas, apesar de serem implementados em diferentes módulos (pacotes).

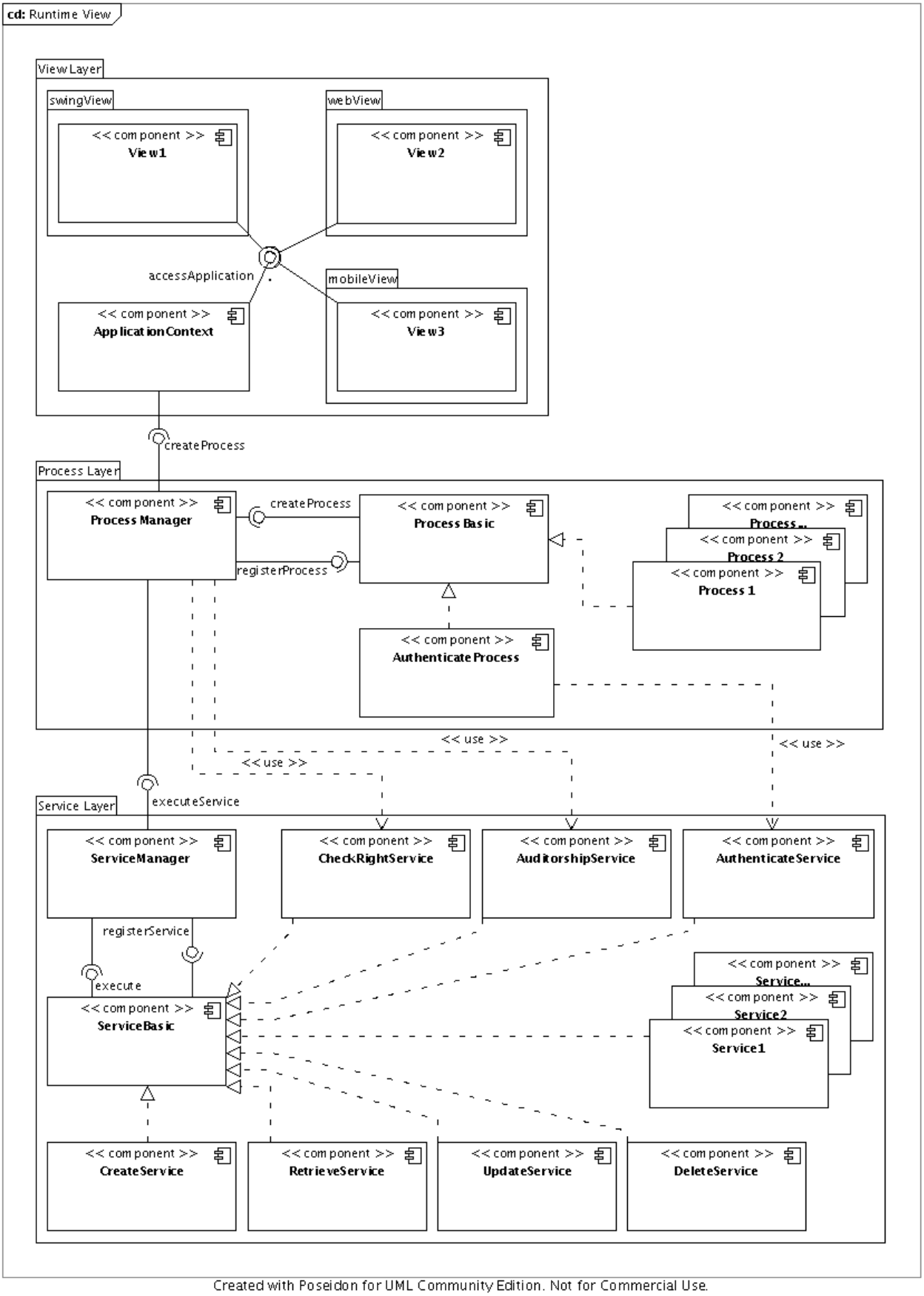


Figura 3.4: Visão de execução da arquitetura de referência (*Runtime View*).

A Tabela 3.4 descreve os componentes da visão de execução.

Tabela 3.4: Descrição dos componentes da visão de execução.

Componente	Descrição
View Layer	Camada lógica onde são alocados os componentes responsáveis pelo controle de interface e acesso ao negócio. Esta e outras camadas são chamadas de camada lógica, porque durante o tempo de execução os componentes que compõem as camadas são integrados dinamicamente. No entanto, podem estar implementadas em diferentes pacotes.
Process Layer	Camada lógica onde são alocados os processos de negócio. O único meio de acesso aos componentes desta camada é o componente *ProcessManager*.
Service Layer	Camada lógica onde são alocados os serviços de negócio. O componente *ServiceManager* é o responsável por gerenciar o acesso a esta camada.
swingView, *webView* e *mobileView*	Estes pacotes representam os diferentes controladores de interface que podem ser implementados e integrados com a arquitetura utilizando o componente *ApplicationContext*.
Application Context	Este componente é responsável por instanciar e interligar todos os demais componentes que se integram à arquitetura. Um arquivo XML é utilizado para a definição dos relacionamentos entre os componentes e alguns parâmetros de instanciação.
Process Manager	Uma interface *registerProcess* é disponibilizada pelo gerenciador para os processos se registrarem. Desta forma, o gerenciador conhecerá todos os processos da aplicação e poderá atender às solicitações da camada de interface quando esta requisitar a instanciação de um determinado processo. Quando um processo é solicitado, o gerenciador executa o serviço *CheckRightService* para verificar se o usuário que solicitou o processo possui direito de acesso ao mesmo. Possuindo o direito necessário, uma instância do processo é criada e sua referência é repassada para o solicitante, que passa a interagir diretamente com o processo. Cada solicitante possuirá sua própria instância de um processo.
Process Basic	Implementa as operações básicas de todo processo que pretende se integrar na arquitetura. Desta forma, um processo pode estender este componente e implementar suas funcionalidades particulares sem se preocupar com as rotinas de integração.
Process 1 Process 2, Process...	Representam os inúmeros processos de negócio que podem ser implementados e integrados à arquitetura. Uma vez integrados, estes processos podem utilizar os componentes da arquitetura, como também serem usados pela arquitetura.
Authenticate Process	Este processo é responsável por verificar se um usuário e senha são válidos. Quando o usuário é autenticado ele recebe uma seção que armazena as informações de autenticação que nos próximos acessos ao gerenciador são repassadas para verificação dos seus direitos. Para validar a identificação do usuário (*login*) e a senha, o processo executa o serviço de autenticação (*AuthenticateService*).
Service Manager	Este componente é responsável pelo gerenciamento dos serviços disponíveis e pela execução dos mesmos. Enquanto o gerenciador de processo é autenticado, o gerenciador de serviços cria um ambiente transacional e executa o serviço solicitado dentro de uma transação. Outra diferença entre este gerenciador e o de processos é que o solicitante do serviço não possui referência ao serviço executado. Para a execução de um determinado serviço, uma estrutura de parâmetros é preenchida, onde é informado o identificador do serviço que deverá ser executado. Maiores detalhes sobre esta estrutura do gerenciador de serviços são encontrados no item 4.2.2. O gerenciador então identifica o serviço, executa o serviço, monitora sua execução, obtém o resultado e o repassa para o solicitante. Por questões de desempenho, é opcional a execução de um serviço dentro de uma

	transação. Isto pode ser definido no arquivo de configuração do componente *applicationContext*.
Service Basic	Implementa as operações básicas de todo serviço que pretende se integrar na arquitetura. Desta forma, um serviço pode estender este componente e implementar suas funcionalidades particulares sem se preocupar com as rotinas de integração.
CheckRight Service	Este serviço implementa as rotinas que acessam as entidades de segurança e verifica se o usuário possui ou não direitos de acesso ao recurso solicitado, também passado como parâmetro para o serviço.
AuditorShip Service	Os processos podem registrar informações sobre suas execuções utilizando este serviço. Desta forma, consultando os registros da auditoria é possível identificar quais processos foram executados, por quais usuários, em quais estações de trabalho e com quais parâmetros.
Authenticate Service	Este serviço valida a identificação de usuário e senha fornecidos. As entidades de segurança são consultadas por este serviço para esta validação.
Service 1, Service 2, Service...	Estes serviços representam os serviços que podem ser implementados e integrados à arquitetura. É importante destacar que para esta integração os serviços estendem um componente básico chamado *ServiceBasic*. Este componente implementa as funcionalidades básicas de integração. Um serviço não possui referência para outro serviço, todo acesso, até mesmo entre serviços, é realizado utilizando o *ServiceManager*.
CreateService, RetrieveService, UpdateService e DeleteService	Estes serviços fazem parte do módulo *crud*. Eles implementam as rotinas para criação, obtenção, atualização e remoção de entidades persistidas pela arquitetura.

Na visão de execução são mostrados alguns pontos que já dão noção de como a arquitetura de referência pode ser estendida. Estendendo os componentes básicos é possível integrar novas funcionalidades à aplicação sem muita preocupação com a instanciação destas novas funcionalidades e sua integração com o restante da arquitetura. Esta estrutura de integração não somente restringe a comunicação entre um determinado componente e outro, mas proporciona um maior entendimento sobre todas as funcionalidades da aplicação, uma vez que define um padrão que é seguido por toda a arquitetura.

A próxima visão na Figura 3.5, a de implementação, mostra como os componentes que implementam a arquitetura de referência podem ser integrados a uma aplicação para execução na Internet. Nesta visão, a arquitetura de referência é implementada em Java e o resto da aplicação utiliza a tecnologia J2EE[1]. As partes em laranja mostram o que deve ser implementado pelo usuário da arquitetura. As partes em verde representam os componentes da arquitetura de referência que são usados pelo resto da aplicação. Nesta visão, que é baseada

[1] Plataforma de desenvolvimento de aplicações corporativas mantida pela Sun (http://java.sun.com/javaee/).

na UML2.0, foi utilizado o recurso de estereótipos para melhor identificar e classificar os componentes. Estes estereótipos são listados na Tabela 3.5.

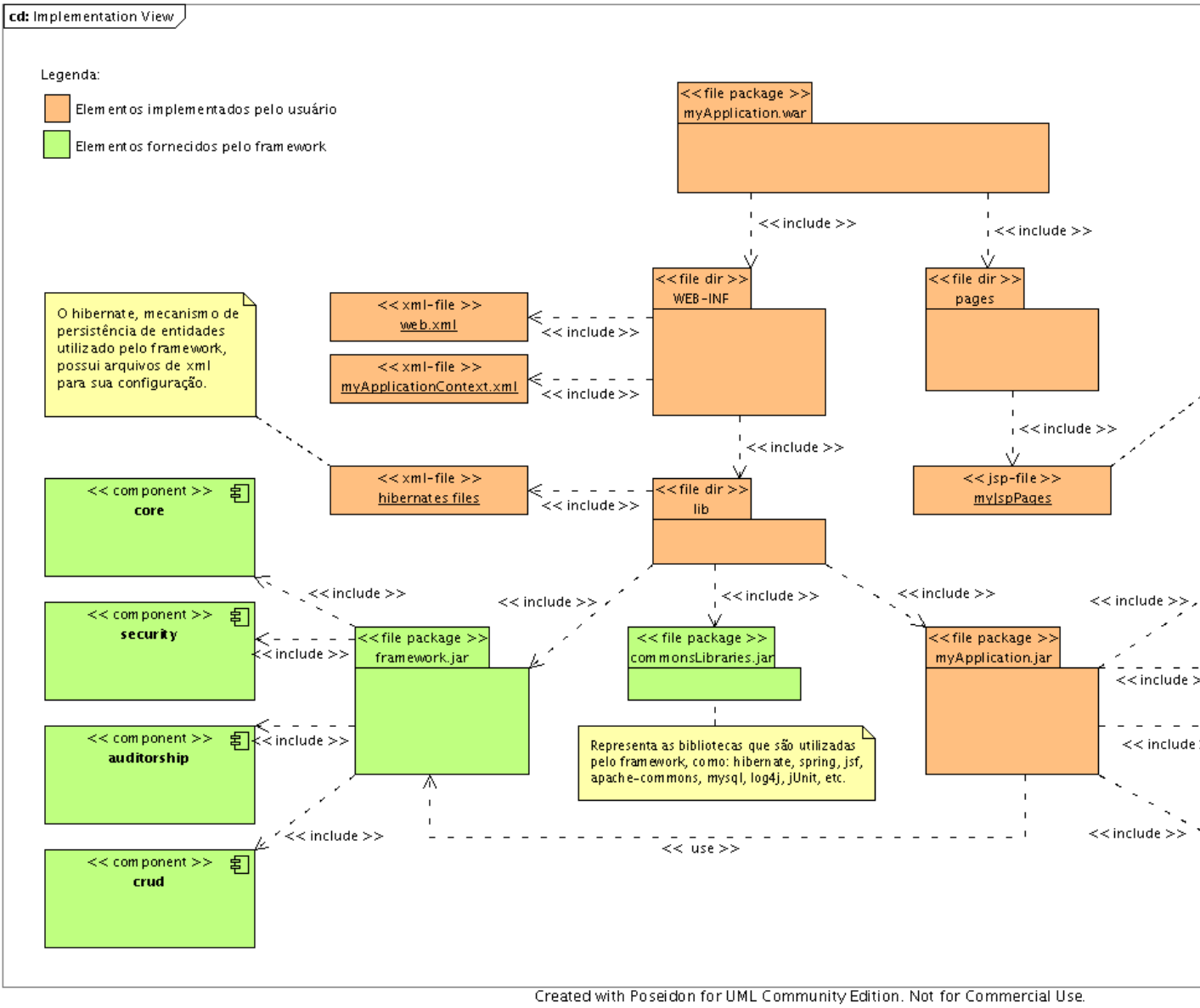


Figura 3.5: Visão de implementação da arquitetura de referência.

Tabela 3.5: Estereótipos utilizados na visão de implementação.

Estereótipo	Descrição
<<file package>>	Identifica os arquivos que empacotam diretórios (pastas) e arquivos.
<<file dir>>	Representa os diretórios no sistema de arquivo ou dentro de arquivos empacotadores.
<<xml-file>>	Identifica os arquivos no formato XML[1].
<<jsp-file>>	Representa os arquivos utilizados na composição de páginas dinâmicas na Internet. Estes arquivos usam a tecnologia JSP[2].

Nesta visão de implementação são representados alguns padrões definidos pela tecnologia J2EE. A estrutura de diretórios e o conteúdo de cada diretório são especificados pela tecnologia. Seguindo este padrão, a aplicação poderá ser instalada em diversos servidores de aplicação que utilizam o padrão J2EE.

A Tabela 3.6 apresenta a descrição dos elementos que compõem a visão de implementação da arquitetura de referência (Figura 3.5). Estes elementos são classificados como: arquivos, pacotes de arquivos, diretórios e componentes de software.

Tabela 3.6: Descrição dos elementos da visão de implantação.

Arquivo	Descrição
web.xml	Este arquivo é utilizado pelo servidor de aplicação para configurar, instanciar e disponibilizar acesso à aplicação executada.
myApplicationContext.xml	Este arquivo é utilizado pela arquitetura para definir a integração dos componentes. Várias aplicações podem usar os mesmos componentes com diferentes configurações para cada necessidade da aplicação. Esta configuração é realizada neste arquivo.
hibernate files	Representa os arquivos de configuração do mecanismo de persistência de objetos que é utilizado na aplicação mostrada por esta visão. O Hibernate[3] foi escolhido para a implementação do protótipo deste trabalho. Maiores detalhes sobre a integração deste mecanismo e a arquitetura são encontrados no capítulo 4.
myJspPages	Estes são os arquivos das páginas da aplicação que são mostradas para o usuário.
Pacote de Arquivos	Descrição
framework.jar	Pacote de arquivos que contém os componentes da arquitetura de referência, implementados na linguagem Java.
commonsLibraries.jar	Pacote com os arquivos das bibliotecas de componentes de terceiros que são utilizados pelo *framework.jar*, ou ainda, pela aplicação do usuário.
myApplication.jar	Este pacote de arquivo representa os componentes que foram implementados utilizando a arquitetura de referência, por meio da extensão dos componentes básicos.

[1] http://www.w3c.org/XML

[2] http://java.sun.com/products/jsp/

[3] http://www.hibernate.org

myApplication.war	Este pacote de arquivos empacota todos os demais elementos de implantação em um único arquivo. Este arquivo é entregue ao servidor de aplicação que conhece seu formato interno e instancia a aplicação.
Diretório	Descrição
WEB-INF	Este diretório é especificado pelo padrão J2EE para conter os arquivos de configuração da aplicação.
pages	Este diretório armazena todas as páginas e subdiretórios da aplicação que são visíveis aos navegadores WEB.
lib	Este diretório é especificado pelo padrão J2EE para conter todos os arquivos de bibliotecas que serão necessários durante a execução da aplicação.
Componente	Descrição
core	Este componente representa o núcleo básico de implementação da arquitetura de referência. Este núcleo é composto pelos gerenciadores das camadas de visualização, processos e serviços, e ainda, pelas classes básicas que permitem a integração das classes do usuário com a arquitetura. Maiores detalhes sobre este núcleo são encontrados no capítulo 4.
security	Este componente implementa as funcionalidades de segurança da arquitetura de referência. Maiores detalhes sobre este componente são encontrados no capítulo 4.
auditorship	Este componente implementa as funcionalidades de auditoria da arquitetura de referência. Maiores detalhes sobre este componente são encontrados no capítulo 4.
crud	Este componente implementa o mecanismo de abstração e persistência de entidades da aplicação. Maiores detalhes sobre este componente são encontrados no capítulo 4.
myJspViews	Este componente representa os componentes que o usuário implementa para controlar as visualizações de sua aplicação utilizando a tecnologia JSP.
myProcesses	Representa a implementação dos processos do usuário.
myServices	Representa a implementação dos serviços do usuário.
mySupportComponents	Representam outros componentes que o usuário implementa que dão algum suporte à execução de seus serviços.

A Figura 3.6 apresenta a visão de como a aplicação é implantada no ambiente de execução. Tratando-se da implementação de uma aplicação para ser acessada pela Internet, é utilizado um servidor de aplicação no qual a aplicação é instalada. Este servidor é responsável por instanciar a aplicação e gerenciar o acesso da mesma pelos usuários da Internet. A aplicação é acessada através de um navegador de páginas da Internet. As páginas são geradas pela tecnologia JSP utilizada na aplicação.

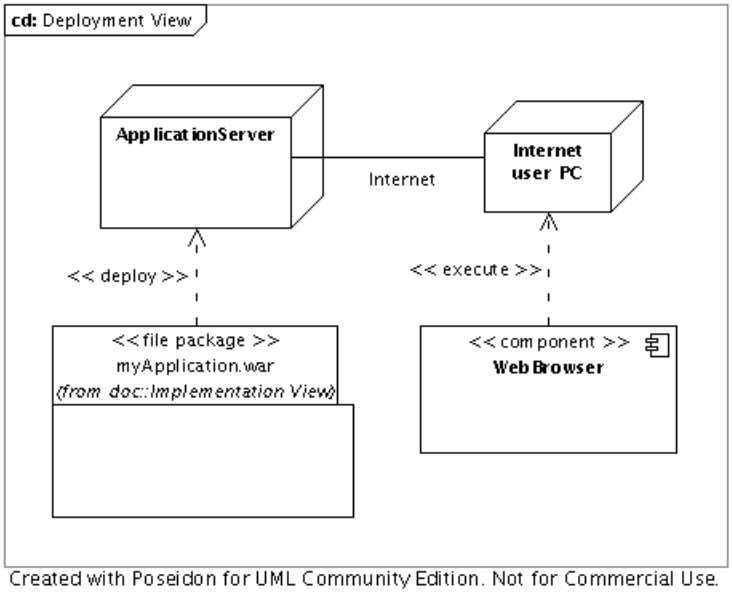


Figura 3.6: Visão de instalação.

Os próximos diagramas (Figura 3.7 e Figura 3.8) mostram uma visão do comportamento da arquitetura. Nestes dois diagramas é mostrado como um usuário se autentica na arquitetura e a partir daí começa a invocar outros processos de negócio baseando em seus direitos de acesso.

A autenticação de um usuário deve ser realizada utilizando uma interface de interação que solicita o nome e a senha do usuário. Na Figura 3.7 é mostrada a interação desta interface com o restante da arquitetura. Esta interface possui uma referência ao contexto da aplicação e, assim, consegue acessar todo o resto da aplicação. Acessando o contexto da aplicação (*Application Context*), a interface obtém uma referência do gerenciador de processos (*ProcessManager*) e, então, obtém uma instância do processo de autenticação de usuários (*AuthenticateProcess*).

Interagindo com o processo, a interface envia o nome e a senha do usuário para o processo e solicita a autenticação. Ao concluir a autenticação com sucesso, o processo cria uma seção com as informações de autenticação. Esta seção é solicitada pela interface para que seja armazenada e utilizada nas próximas invocações de processos. O processo de autenticação não é um processo autenticado, ele é um processo público, ou seja, qualquer usuário pode acessá-lo. Os processos públicos são executados pela arquitetura sem a exigência de uma seção autenticada, ou seja, sem a identificação do usuário que está solicitando. Este tipo de processo é útil para a implementação de funcionalidades acessíveis a qualquer usuário da aplicação.

O segundo diagrama (Figura 3.8) mostra como o gerenciador de processo utiliza o serviço de verificação de direitos do usuário (*CheckRightService*) para determinar se um usuário tem ou não direito de acesso a determinado processo.

Com a apresentação das visões dos módulos de execução, de implementação, de implantação e de comportamento, encerra-se a seção de documentação da arquitetura de referência.

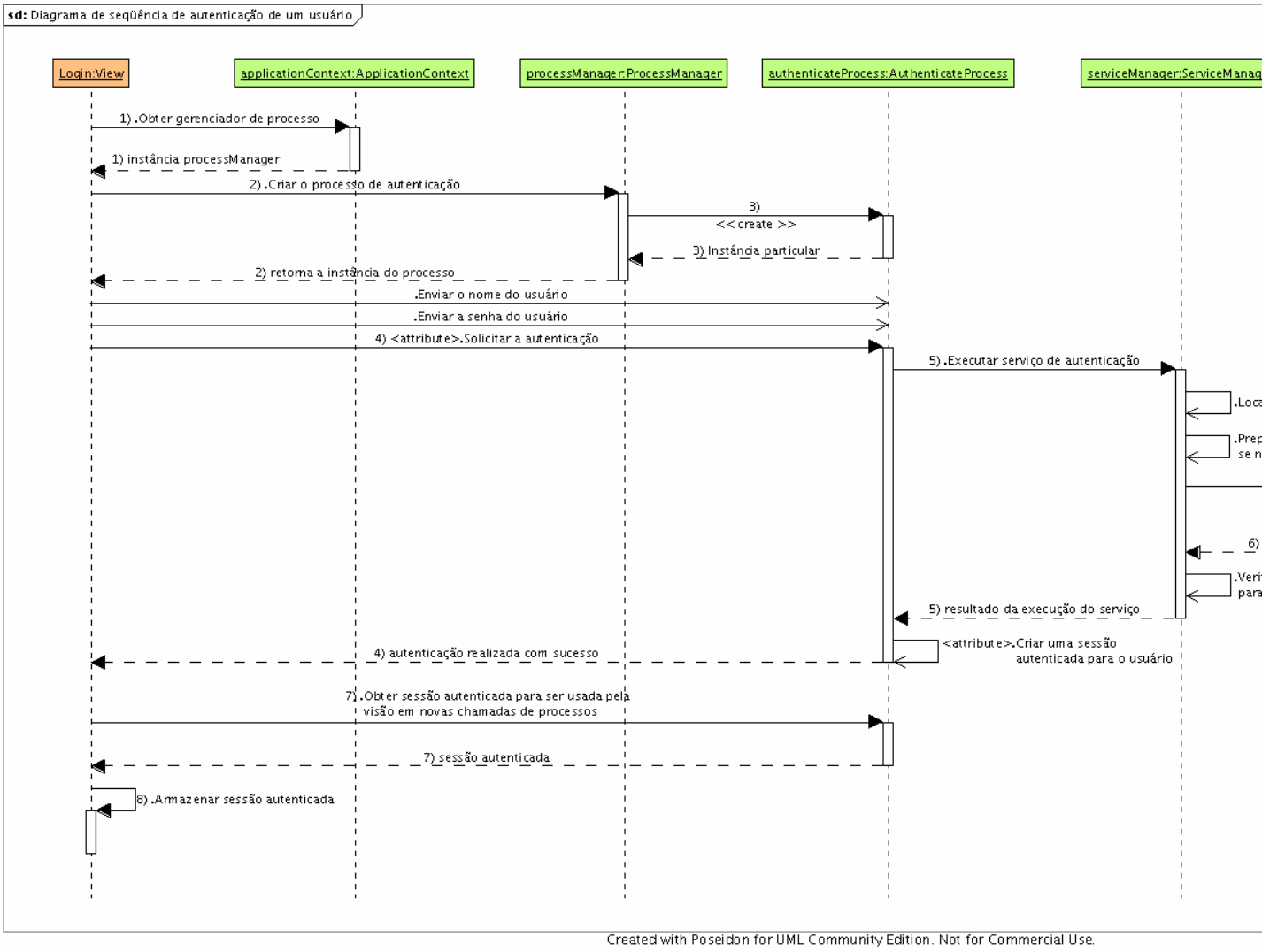


Figura 3.7: Diagrama de seqüência de autenticação de um usuário na arquitetura de refe

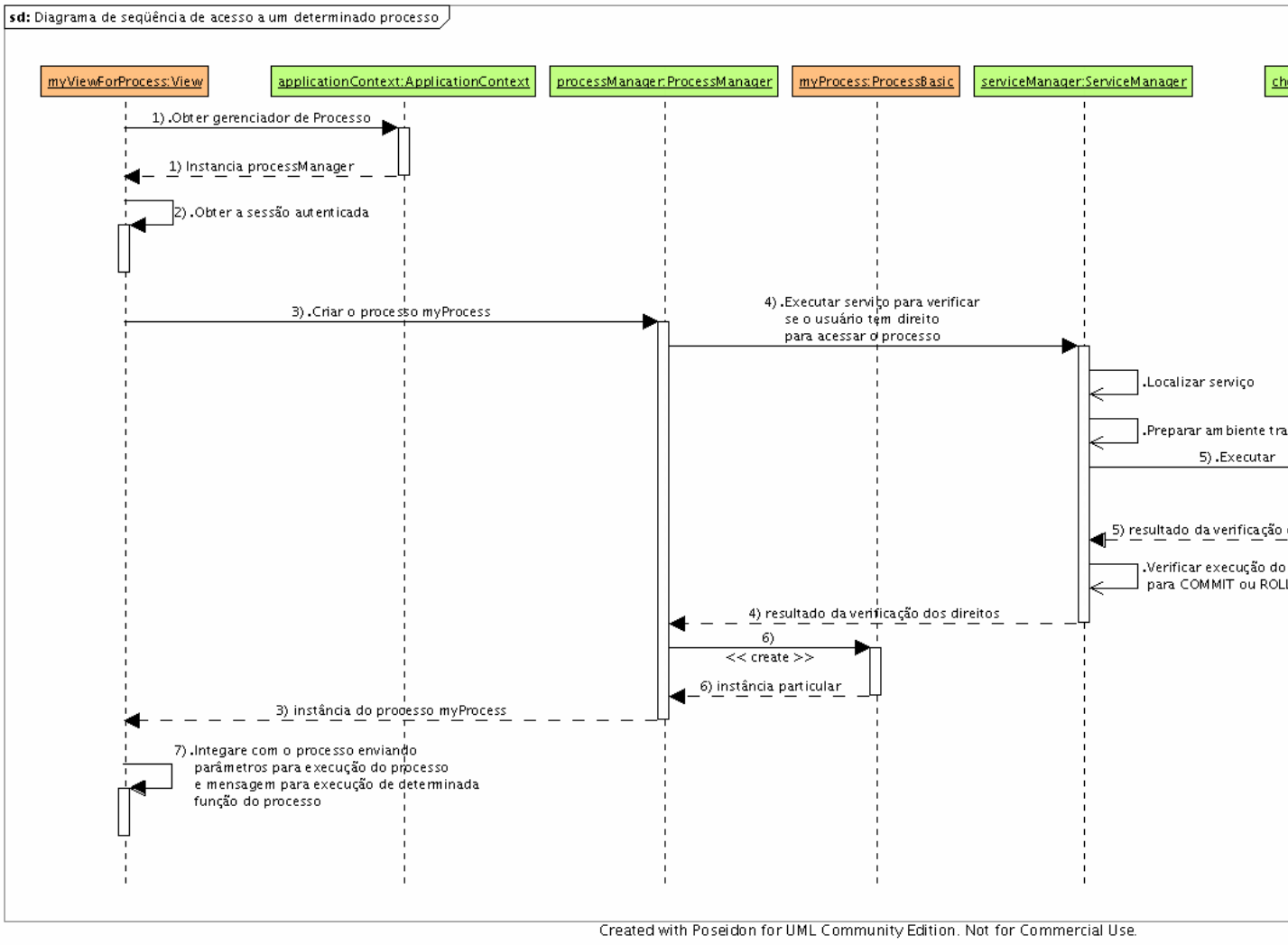


Figura 3.8: Diagrama de seqüência para execução de um processo por um usuário autor

3.4. Arquitetura de software

Seguindo o esquema apresentado por Bass et al (2003), mostrado na Figura 2.10, e concluída a arquitetura de referência, o próximo passo foi definir a arquitetura de software, que é específica para o domínio KDD. Nesta seção são apresentados a arquitetura de software e os passos seguidos para a sua composição. Para apresentação da arquitetura de software foi utilizada a visão de módulos e a análise dos casos de usos do sistema KDD.

3.4.1. Visão de módulos

Seguindo o modelo de referência, apresentado no início deste capítulo, foram definidos os módulos: *source*, *etDefinition*, *staging*, *loadDefinition*, *dw*, *olap*, *dm*, *result* e *viewTools*, representados na Figura 3.9.

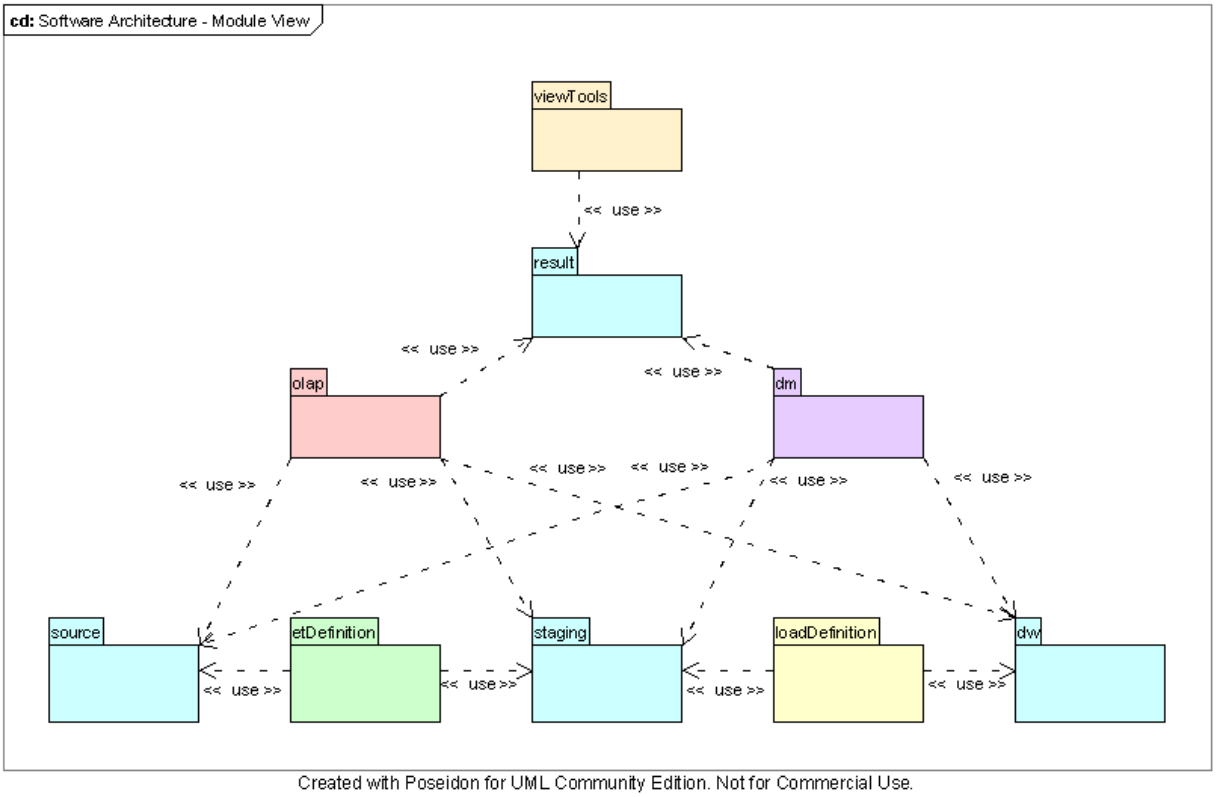


Figura 3.9: Visão dos módulos da arquitetura de software do sistema KDD.

Com exceção das bases de dados, foi definido um módulo para cada elemento apresentado no modelo de referência. Esses módulos contêm as funcionalidades básicas para a realização das atividades do sistema KDD. Não houve aqui um trabalho exaustivo para uma definição minuciosa de todas as funcionalidades possíveis de um sistema KDD. É mantido o foco na

definição da arquitetura que pode ser utilizada pelas equipes de desenvolvimento de sistemas KDD. Este trabalho deixa a cargo de cada equipe de desenvolvimento de sistema a definição detalhada dos inúmeros recursos para a execução de seu processo KDD.

Além da visão de módulo, foi utilizado um diagrama de casos de uso para detalhar a arquitetura de software. A próxima seção apresenta os casos de usos, suas descrições e os modelos criados para as realizações.

3.4.2. Casos de uso

Para auxiliar no entendimento dos requisitos funcionais e na concepção da arquitetura de software foi utilizado um diagrama de casos de uso, definido pela UML e utilizado para captura de requisitos pelo *Unified Process* (Jacobson e Booch, 1998), que é um processo de desenvolvimento de software guiado por casos de uso e centrado em arquitetura. A Figura 3.10 representa os principais casos de uso que foram identificados no modelo de referência. Esses casos de uso guiaram o desenvolvimento do protótipo para validação da arquitetura proposta por este trabalho. A descrição dos atores e dos casos de uso é apresentada na Tabela 3.7.

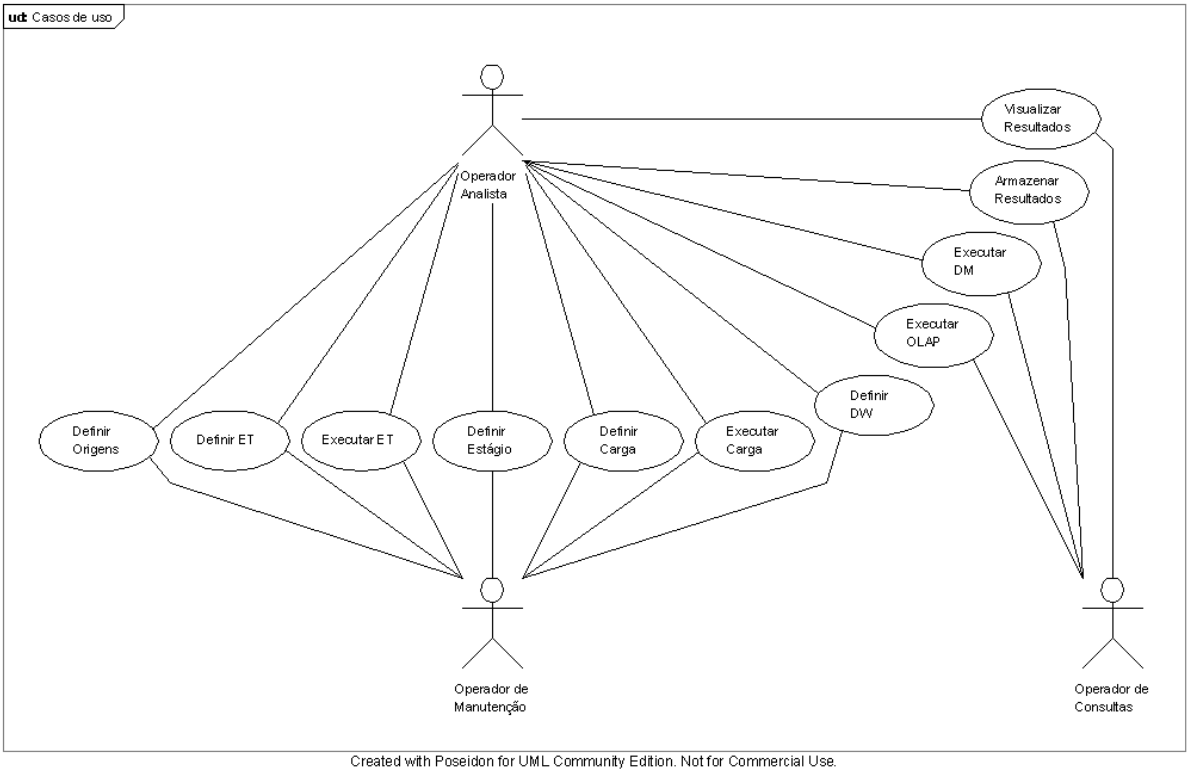


Figura 3.10: Casos de uso do sistema KDD representado pelo modelo de referência.

Tabela 3.7: Descrição dos atores e casos de uso do sistema KDD.

Ator	Descrição
Operador Analista	Este ator desempenha o papel do analista de busca de conhecimento. Ele interage com o sistema para definir como serão realizadas as atividades do seu processo de KDD utilizando o sistema. É necessário que ele tenha acesso a todo o sistema para executar algumas atividades e analisar os resultados obtidos a fim de decidir sobre as configurações do seu processo.
Operador de Manutenção	Este ator representa os usuários que utilizam o sistema para aplicar as definições de processos KDD que foram previamente analisados e tiveram todas as suas estruturas definidas por meio de documentos criados pelo analista.
Operador de Consultas	Este ator representa os usuários que utilizam o sistema somente para executar operações de OLAP ou de MD, armazenar os resultados e visualizar os resultados armazenados. Dependendo do nível de complexidade das interfaces criadas pela equipe de desenvolvimento do sistema KDD, este ator pode representar um gerente do negócio ou um usuário mais técnico em função do gerente.
Caso de uso	Atividades
Definir Origens	• O sistema permite manter as conexões para acesso aos gerenciadores de banco de dados de origem: Esta atividade cria, atualiza ou exclui os parâmetros das conexões de origem. Esses parâmetros permitem identificar o endereço de rede dos gerenciadores, a tecnologia de armazenamento e as informações de autenticação de usuário para o acesso dos dados. • O sistema lista as conexões mantidas; • O ator seleciona uma conexão para que o sistema liste as tabelas da conexão selecionada; • O ator seleciona uma tabela para que o sistema liste os itens de dado da tabela selecionada pelo usuário; • O ator marca os itens de dado da tabela que estarão disponíveis para o sistema KDD; • O ator confirma a seleção; • O sistema armazena os itens selecionados em estruturas de controle internas que descrevem as bases de origens. Estes itens de dado estarão disponíveis para as outras atividades do sistema KDD quando estas consultarem o que pode ser acessado nas bases de origens.
Definir ET	• O sistema lista os itens de dado disponíveis na área de origem; • O ator seleciona um item que pode ser acessado na base de origem; • O ator define as propriedades de agendamento da extração e transformação; • O ator define as funções e parâmetros de transformação que serão aplicadas no item de dado selecionado, juntamente com os parâmetros;

	• O sistema lista os itens de dado da área de estágio • O ator seleciona um item de dado da área de estágio para o *estagiamento*[1] do dado transformado; • O ator confirma a definição; • O sistema armazena as definições de extração, transformação e estágio em suas estruturas internas;
Executar ET	• O sistema lista as definições de extração, transformação e estágio. Para isto, utiliza um filtro que mostra somente as definições cujas propriedades de agendamento disponibilizam sua execução para a data atual; • O ator marca quais definições deseja executar; • O ator confirma a execução; • O sistema executa as definições e informa o estado atual de execução e de sua finalização com sucesso ou com falha. Observação: Aqui há dois pontos para serem explorados em trabalhos futuros: o primeiro é a extensão deste caso de uso para que ele possa ser executado em segundo plano, ou seja, uma vez disparada sua execução, ela continua sem a intervenção do ator até seu término, quando o sistema avisa o ator que uma de suas execuções foi concluída. Outro ponto é a definição de um agente que execute automaticamente as definições programadas sem nenhuma intervenção, tornando automático o processo de extração, transformação e estágio.
Definir Estágio	• O sistema permite manter as conexões para acesso à área de estágio: Todas as conexões seguem o mesmo padrão de manutenção definido no caso de uso *Definir Origens*, na atividade de manter as conexões; • O sistema permite manter as tabelas e os itens de dado da área de estágio; • O sistema permite sincronizar as definições da área de estágio com as conexões da área de estágio: Esta atividade faz com que as definições de tabelas e de itens de dado da área de estágio, que são persistidas no sistema, sejam sincronizadas com o esquema de dados que se encontra no gerenciador de banco de dados acessados pela conexão. Desta forma, a criação, alteração ou exclusão de alguma tabela ou item de dado, realizada pelo usuário, será refletida na tabela ou item de dado que se encontra no gerenciador.
Definir Carga	• O sistema lista as conexões da área de estágio; • O ator pode selecionar uma conexão e analisar sua estrutura interna; • O ator define as propriedades de agendamento de carga. • O ator define as funções e parâmetros de carga; • O ator seleciona uma conexão do DW; • O ator confirma a definição; • O sistema armazena as definições de extração, transformação e

[1] Estagiamento não é uma palavra encontrada nos dicionários da língua portuguesa. Neste trabalho ela é utilizada como neologismo para definir a atividade de armazenamento dos dados extraídos das bases de origem na área de estágio.

	estágio em suas estruturas internas;
Executar Carga	• O sistema lista as definições de carga. Para isto, utiliza um filtro que mostra somente as definições cujas propriedades de agendamento disponibiliza sua execução para a data atual; • O ator marca quais definições deseja executar; • O ator confirma a execução; • O sistema executa as definições e informa o estado atual de execução e de sua finalização com sucesso ou com falha. • Observação: Aqui também há pontos para serem explorados em trabalhos futuros e automatizar este caso de uso para diminuir a interação do ator.
Definir DW	• O sistema permite manter as conexões para acesso ao DW: Todas as conexões seguem o mesmo padrão de manutenção definido no caso de uso *Definir Origens*, na atividade de manter as conexões; • O sistema permite manter os fatos e as dimensões do DW; • O sistema permite sincronizar as definições do DW com as conexões do DW: Esta atividade tem o mesmo objetivo da atividade de sincronização definida no caso de uso *Definir Estágio*. A diferença é que no DW há tabelas de fatos e de dimensões.
Executar OLAP	• O sistema disponibiliza recursos para execução de operações de OLAP. • O ator executa as operações; • O ator pode armazenar os resultados das operações.
Executar DM	• O sistema disponibiliza recursos para execução de operações de MD. • O ator executa as operações; • O ator pode armazenar os resultados das operações.
Armazenar Resultados	O sistema disponibiliza recursos para que os resultados de OLAP e de MD possam ser armazenados em um repositório e possam ser utilizados por ferramentas de visualização ou ainda, reutilizados em novos processos de OLAP e de MD.
Visualizar Resultados	Este caso de uso envolve a integração de ferramentas de visualização com o sistema KDD. Essas ferramentas possibilitam que os resultados de OLAP e de MD sejam apresentados em gráficos, tabelas, ou ainda, em qualquer outra forma que facilite a compreensão dos resultados obtidos.

Seguindo a arquitetura de referência, cada módulo define suas visualizações, seus processos, seus serviços e seus componentes de apoio. A arquitetura de software é definida estendendo as funcionalidades encontradas na arquitetura de referência. Lembrando que na arquitetura de referência foram definidas algumas classes básicas que servem como pontos de integração de componentes externos com a arquitetura e seus componentes internos.

A arquitetura de software é uma arquitetura específica para o domínio KDD, ou seja, os componentes por ela definidos estão diretamente ligados ao domínio do sistema KDD,

podendo, no entanto, não ser exclusivos deste domínio. Como a arquitetura de referência resolve a maioria dos problemas estruturais, a definição da arquitetura de software é facilitada e seus componentes são abstraídos dos conceitos definidos pela arquitetura de referência. Assim, a arquitetura de software do sistema KDD apresentada neste trabalho consiste na realização dos casos de uso que foram identificados na Figura 3.10. Para esta realização é utilizado o exemplo ilustrativo mostrado na seção 3.2 (*A escolha do estilo arquitetural*), que mostra como os componentes de cada camada podem ser extraídos de um caso de uso.

A seguir são apresentados os casos de usos e os componentes que os realizam. Cada caso de uso tem sua realização comentada. Para um melhor entendimento, recomenda-se acompanhar os comentários observando os detalhes representados na respectiva figura. Os componentes da camada de visualização, encontrados nos pacotes chamados *View*, são responsáveis pela interação do usuário com o processo em execução, por esta razão, nos comentários sua função é muitas vezes subentendida.

Definir Origens

A Figura 3.11 mostra as entidades, serviços, processos e visualizações que foram obtidas deste caso de uso. As estruturas das bases de origens são definidas pelos sistemas legados, responsáveis por essas bases. Contudo, nesta proposta, para organizar a manipulação dos dados de origens, foi definido que é necessário executar um processo de seleção que define o que pode ser acessado pelo sistema KDD para se ter acesso a esses dados. Essa seleção de tabelas e itens de dados é armazenada em entidades internas do sistema KDD.

A entidade *SourceConnection* armazena as informações de uma conexão com as bases de dados de origem. Cada conexão é relacionada com inúmeras entidades do tipo *SourceTable* que descrevem a estrutura das tabelas de origens. Por sua vez, cada entidade *SourceTable* possui inúmeras entidades *SourceField* que descrevem os itens de dado das tabelas de origens.

Para a manipulação das entidades representadas na Figura 3.11 e a realização das atividades de definição das origens foram criados dois processos:

- *CreateConnectionProcess:* Controla a criação de uma nova conexão, solicitando as informações necessárias, validando essas informações e as persistindo quando confirmadas. Este processo realiza as atividades de manutenção de conexões do caso de uso *Definir Origens*.

- *DefineSourcePropertiesProcess*: Controla a definição das tabelas e dos itens de dado das bases de dados de origens a se tornarem acessíveis pelo sistema KDD. Este processo realiza as atividades de listagem e de definição do caso de uso *Definir Origens*. Para listar as conexões mantidas pelo sistema KDD são utilizados os serviços do módulo CRUD que manipula as entidades das conexões. Os serviços *ListTablesService* e *ListTableFieldsService* são usados para listar as tabelas de uma conexão e os itens de dado de uma tabela, respectivamente.

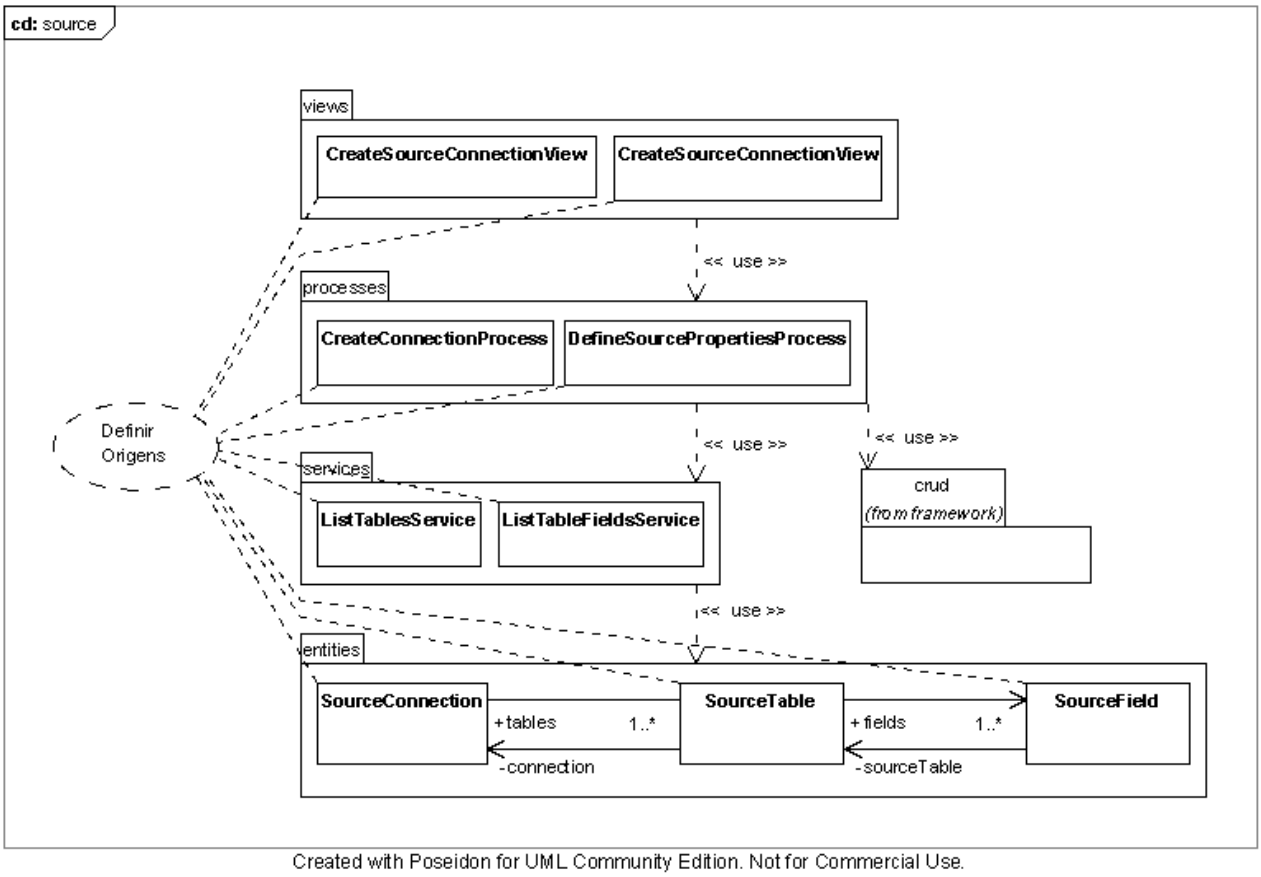


Figura 3.11: Entidades, serviços, processos e visualizações definidas pelo caso de uso *Definir Origens*.

Definir ET

A Figura 3.12 mostra a interação entre os elementos definidos para a realização do caso de uso que define as transformações para o *estagiamento* dos dados extraídos das bases de origem. Este caso de uso utiliza o módulo *CRUD* para realizar a manutenção de suas entidades. Por isto, não foram definidos processos e serviços para manter estas definições. Como origem dos dados a serem transformados são utilizadas as entidades do tipo *SourceField* que são persistidas no final do caso de uso *Definir Origens* (descrito no item

anterior). Como destino são utilizadas as entidades do tipo *StagingField* que são persistidas pelo caso de uso *Definir Estágio*.

Cada definição de ET, representada pela entidade *EtDefinition*, possui dados que informam a freqüência (entidade *Frequency*) que devem ser executadas, as funções de transformação de dados (entidades *EtFunction)* e os parâmetros que serão aplicados nessas funções (*EtFunctionParam*).

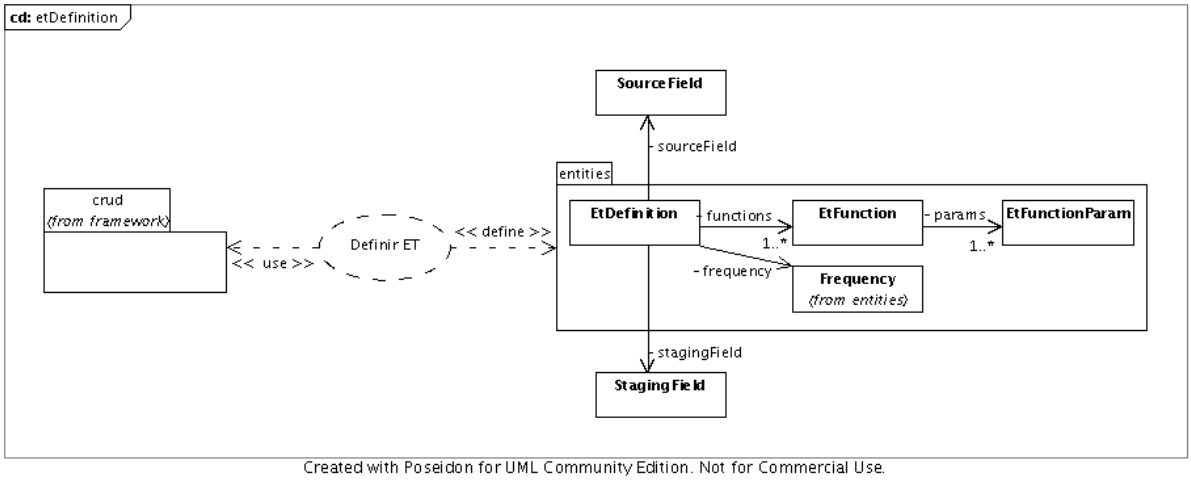


Figura 3.12: Entidades definidas pelo caso de uso *Definir ET* e entidades relacionadas.

Executar ET

Utilizando as informações de freqüência das definições ET (entidade *EtDefinition*), como mostra a Figura 3.13, o serviço *ListEtService* obtém uma lista de definições que podem ser executadas. Este serviço é invocado pelo processo que coordena as atividades de execução de ET, o processo *RunEtProcess*. Depois de obtida, a lista é mostrada para o ator selecionar as execuções desejadas, para isto é utilizada a visualização *RunEtView*. Após a seleção do ator, o processo envia a lista para o serviço *RunEtService* que é responsável por interpretar a definição e executá-las sobre os dados de origem.

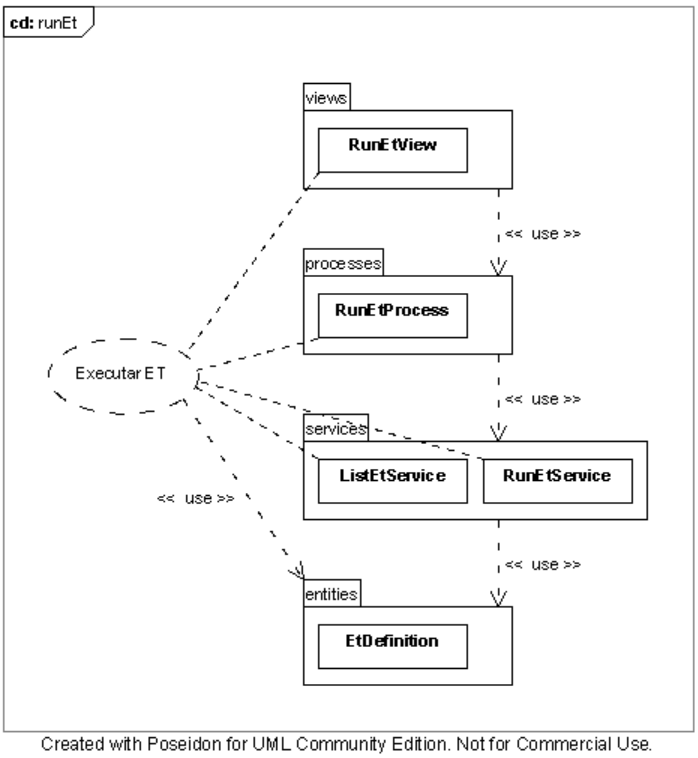


Figura 3.13: Entidade usada, serviços, processo e visualização definidos pelo caso de uso *Executar ET*.

Definir Estágio

Este caso de uso, conforme pode ser visto na Figura 3.14, define as entidades da área de estágio (conexões, tabelas e campos). Para a manutenção dessas entidades é utilizado o módulo CRUD. Além da definição é necessário sincronizar as estruturas da área de estágio com as estruturas armazenadas nos gerenciadores de banco de dados que são apontados pelas conexões. Para esta atividade é definida a visualização *SyncronizeStagingView* por onde o ator inicia a atividade e interage com o processo *SyncronizeStagingProcess*. O serviço *SyncronizeStagingService* é que executa todo o trabalho da sincronização. Após a sincronização, todas as tabelas e campos disponíveis nas conexões da área de estágio estarão consistentes com a estrutura da área de estágio definida.

Pelas breves descrições dos casos de uso até aqui mostradas, pode haver uma dificuldade de entender o papel de cada pacote *view*, *processes* e *sevices*. Para que não haja dúvidas, é importante entender que o processo de sincronização precisa ser iniciado, e isto se dá por alguma ação disparada na camada de visualização que interage com o ator. Esta ação é disparada pela visualização *SyncronizeStagingView*, que depois do disparo fica aguardando um retorno para verificar se realmente o processo pôde ser iniciado corretamente. Em caso de

falha, a visualização já irá identificar a causa e exibir alguma informação para o ator. Se o processo foi iniciado corretamente, então a visualização vai mostrar uma tela para que o ator escolha a conexão de estágio que ele deseja sincronizar com o sistema KDD.

A visualização somente é responsável pela exibição de informações e entrada de dados, ela não tem como, por exemplo, obter a lista de todas as conexões de estágio. Ela não sabe fazer isto, mas ela possui uma referência para o processo que ela manipula. Este processo sim sabe onde obter esta lista. Logo, a visualização somente solicita para o processo esta lista e a exibe. Por sua vez, o processo é programado para utilizar o CRUD e obter esta lista. Depois de listadas as conexões, o ator vai confirmar sua seleção pela visualização. O processo então envia a conexão selecionada para que o serviço *SyncronizeStagingService* identifique o quê deve ser atualizado. Todo o trabalho pesado de processamento fica a cargo do serviço. O processo funciona como um seqüenciador de passos e coletor de parâmetros, enquanto a visualização como um tradutor dos dados que vem do processo para formatos que possam ser exibidos na tecnologia de interface que ela implementa.

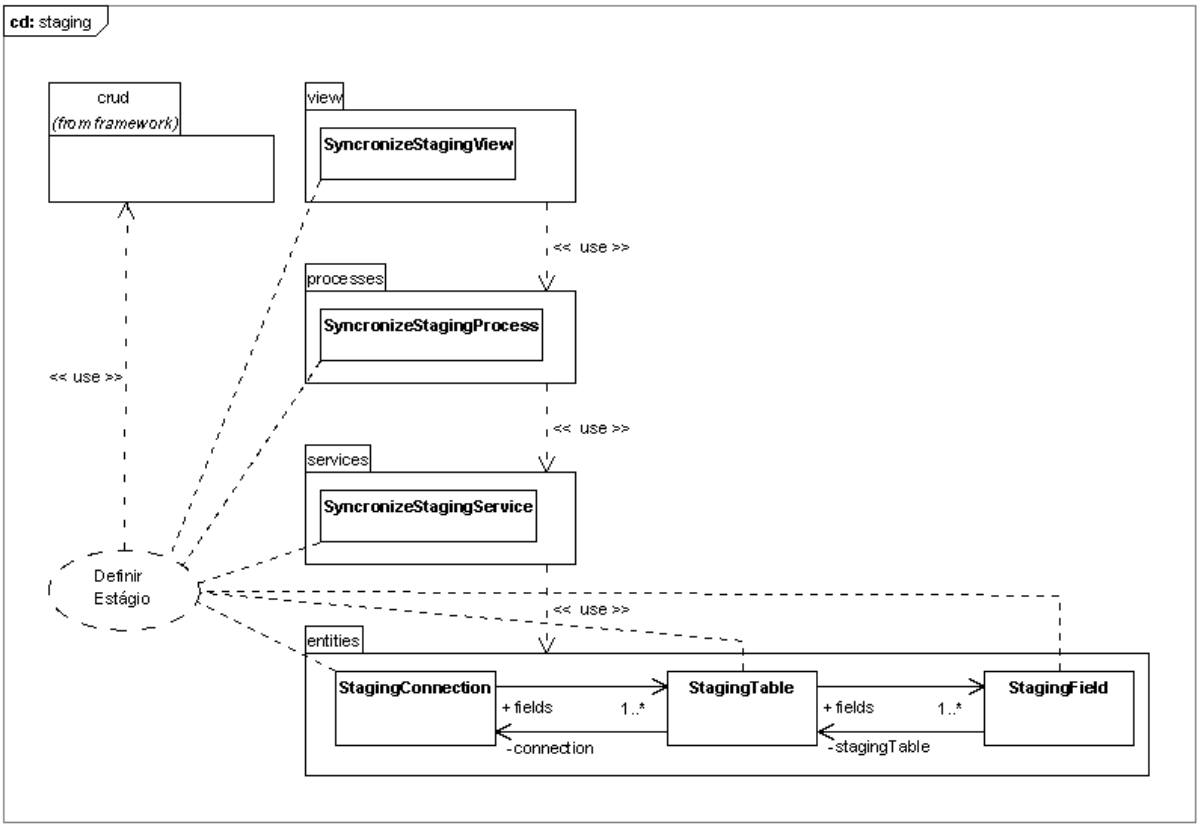


Figura 3.14: Entidades, serviço, processo e visualização obtidos do caso de uso *Definir Estágio*.

Definir Carga

A Figura 3.15 mostra os componentes que realizam este caso de uso. Para a manutenção das entidades é utilizado o módulo CRUD. A entidade *LoadDefinition* possui uma propriedade *frequency* que define as informações sobre a periodicidade de execução de uma definição. Cada definição de carga também se relaciona com uma entidade *LoadFunction* que define as informações sobre a função de carga que será executada sobre os dados. A origem dos dados que serão carregados é a área de estágio cujo acesso é descrito por uma entidade *StagingConnection*. O destino dos dados carregados é o DW que é acessado por uma entidade *DwConnection*.

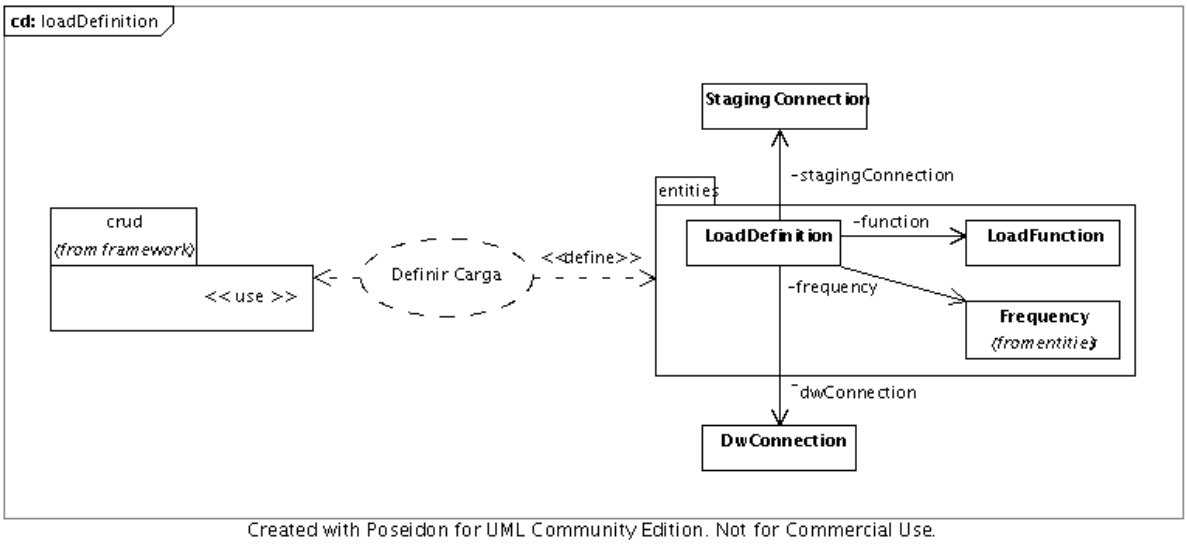


Figura 3.15: Entidades e dependências do caso de uso *Definir Carga*.

Executar Carga

Este caso de uso é mostrado na Figura 3.16 e segue o mesmo padrão do caso de uso *Executar ET*. Ele define uma visualização *RunLoadView*, um processo *RunLoadProcess* e os serviços *ListLoadService* e *RunLoadService*. O serviço *ListLoadService* obtém uma lista de todas as definições de carga que estão disponíveis para execução. O serviço *RunLoadService* é responsável por executar as definições de carga.

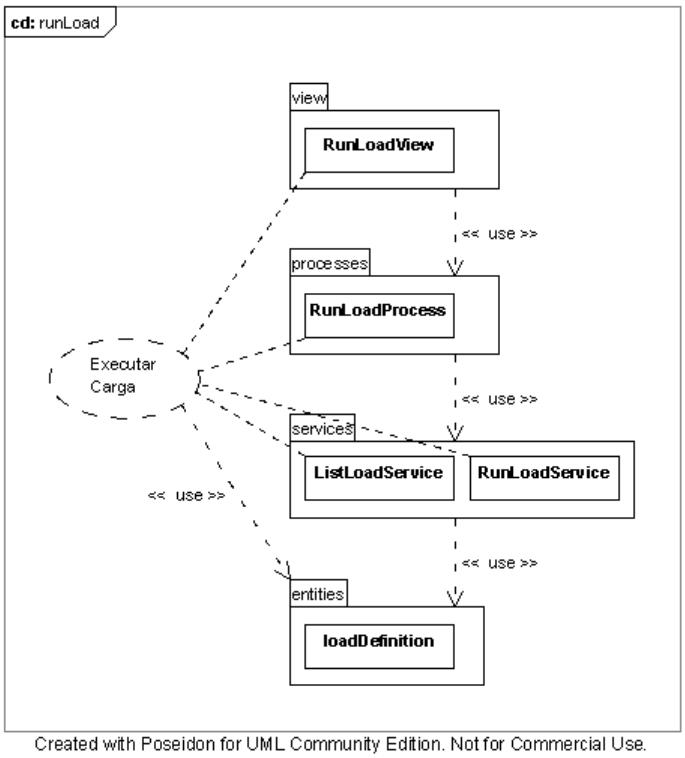


Figura 3.16: Entidade usada, serviços, processo e visualização obtidos do caso de uso *Executar Carga*.

Definir DW

A Figura 3.17 apresenta entidades, serviço, processo e visualização obtidos deste caso de uso. Em um DW é utilizado o modelo dimensional. Neste modelo há tabelas de fatos e de dimensões (Ballard et al., 1998), que na figura são controladas pelas entidades *DwFact* e *DwDimension*, respectivamente. Para cada tabela foram definidas entidades diferentes para controlar a definição dos itens de dado das tabelas de fatos e de dimensões: *DwFactField* e *DwDimensionField*. Quase todos os relacionamentos entre as entidades são bidirecionais. Esta característica é explorada pelo módulo CRUD que permite identificar estes relacionamentos e realizar uma navegação de uma entidade para outra entidade relacionada. Este caso de uso também utiliza o módulo CRUD para manter as suas entidades, o que dispensa a criação de serviços e processos específicos para esta manutenção.

As definições do DW que são persistidas pelas entidades do sistema KDD precisam ser sincronizadas com as estruturas dos gerenciadores de banco de dados onde os dados do DW são armazenados. Para esta atividade foram definidos o serviço *SyncronizeDwService*, o processo *SyncronizeDwProcess* e a visualização *SyncronizeDwView*.

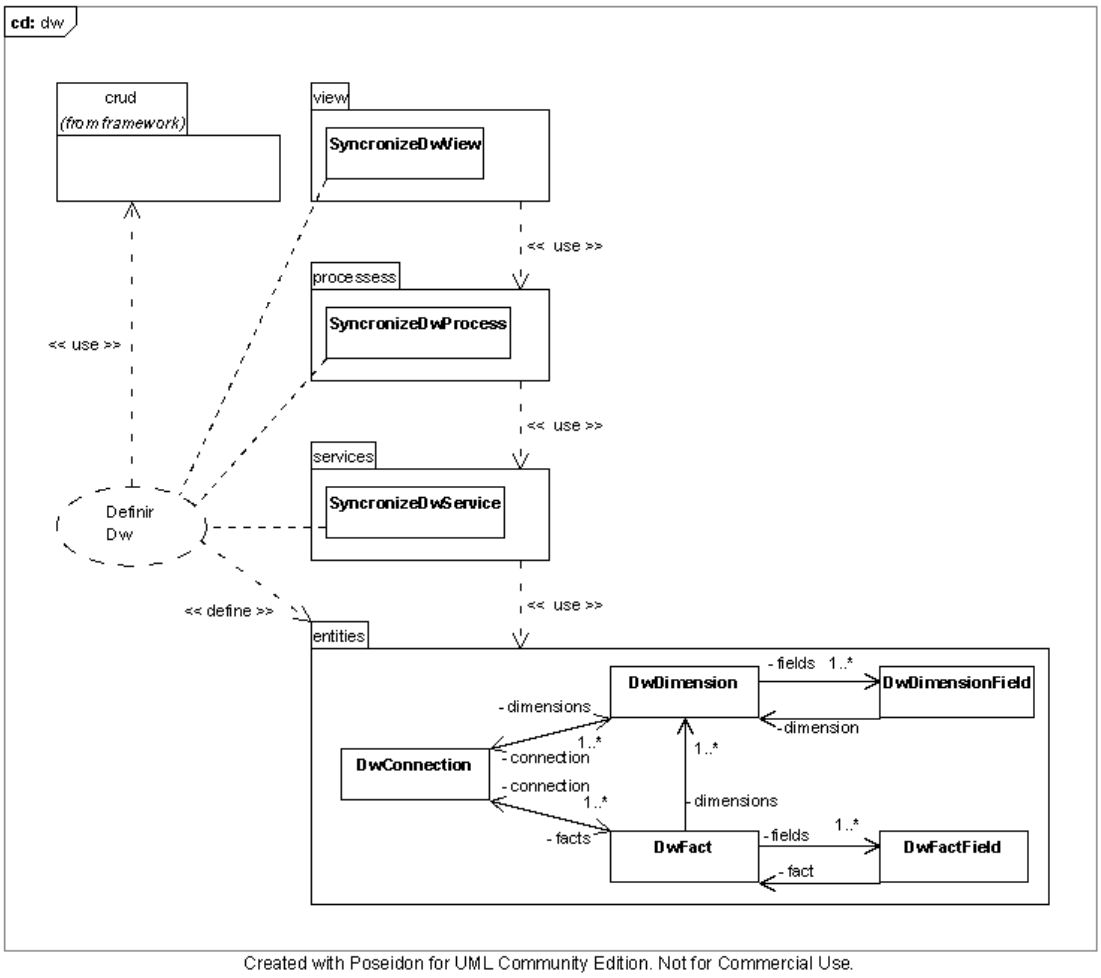


Figura 3.17: Entidades, serviço, processo e visualização obtidos do caso de uso *Definir Dw*.

Armazenar Resultados

A Figura 3.18 mostra as entidades e os serviços definidos por este caso de uso. Este caso de uso é utilizado para armazenar os resultados produzidos pelas operações de OLAP e de MD. Há uma diferença de formatos destes resultados. No atual modelo, as operações de OLAP podem armazenar seus resultados em tabelas e itens de dado (*OlapTable* e *OlapField*). Para as operações de MD foi definida uma entidade *DmResult* que armazena os resultados no formato XML[1]. Isto porque os resultados de uma mineração de dados podem estar em formato de árvore, de lista, binário ou qualquer outro especificado pela técnica de mineração. Desta forma, o padrão XML se apresenta bastante flexível para armazenar qualquer tipo de

[1] http://www.w3.org/XML/

resultado. Os serviços *SaveDmResultService* e *SaveOlapResultService* são responsáveis por receber os resultados de OLAP e de MD e armazená-los adequadamente.

A proposta deste trabalho não restringe a especificação do armazém de resultados, nem de qualquer outro módulo, às atuais estruturas. As estruturas servem somente como um guia inicial para solucionar problemas mais simples. Novas estruturas podem ser criadas, bem como, com a análise de determinada técnica de mineração, a equipe pode decidir por implementar outras estruturas que dão suporte ao armazenamento de seus resultados.

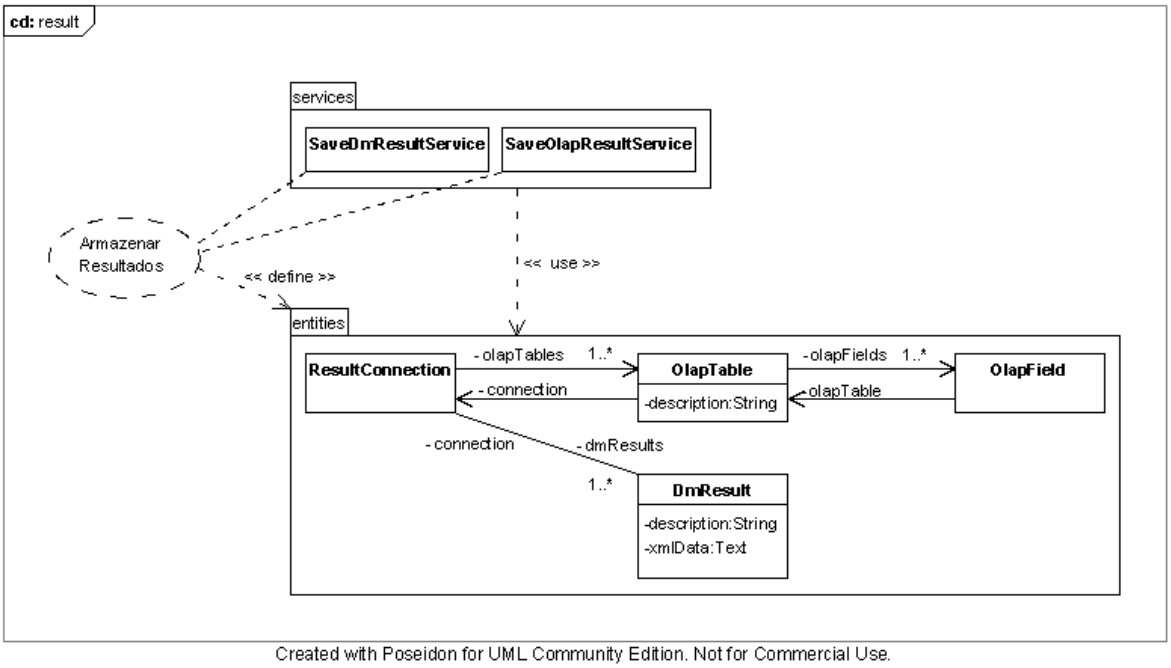


Figura 3.18: Entidades e serviços definidos no caso de uso *Armazenar Resultados*.

Executar OLAP

Neste trabalho, este caso de uso define um serviço *ExecuteOlapService*, um processo *ExecuteOlapProcess* e uma visualização *ExecuteOlapView*. Estes componentes desempenham as funções básicas para execução de consultas OLAP. Mantendo o foco na arquitetura, não foi desenvolvido um estudo mais aprofundado para a especificação de serviços e processos que implementem uma avançada ferramenta OLAP, esta tarefa foi deixada a cargo da equipe de desenvolvedores que utiliza a arquitetura aqui proposta, como também, a definição de serviço que possibilite a integração de uma ferramenta OLAP de sua preferência.

Os casos de uso *Definir Origens*, *Definir Dw* *Definir Estágio* e *Armazenar Resultados* mantêm entidades que armazenam informações sobre conexões de dados disponíveis em cada

repositório. Essas conexões descrevem o que pode ser acessado: tabelas, itens de dado, dimensões e fatos. Com isto, é possível executar operações de OLAP sobre qualquer uma das bases, servindo-se dos dados do sistema KDD em diversos estágios: na origem, transformados na área de estágio, armazenados no DW e armazenados no armazém de resultados. Esta proposta permite que alguns resultados possam ser alcançados sem a conclusão completa de todo o sistema KDD. Por exemplo, com a implementação dos módulos *source* e *olap*, já é possível disponibilizar uma versão do sistema para que os usuários realizem pequenas consultas OLAP diretamente nas bases operacionais.

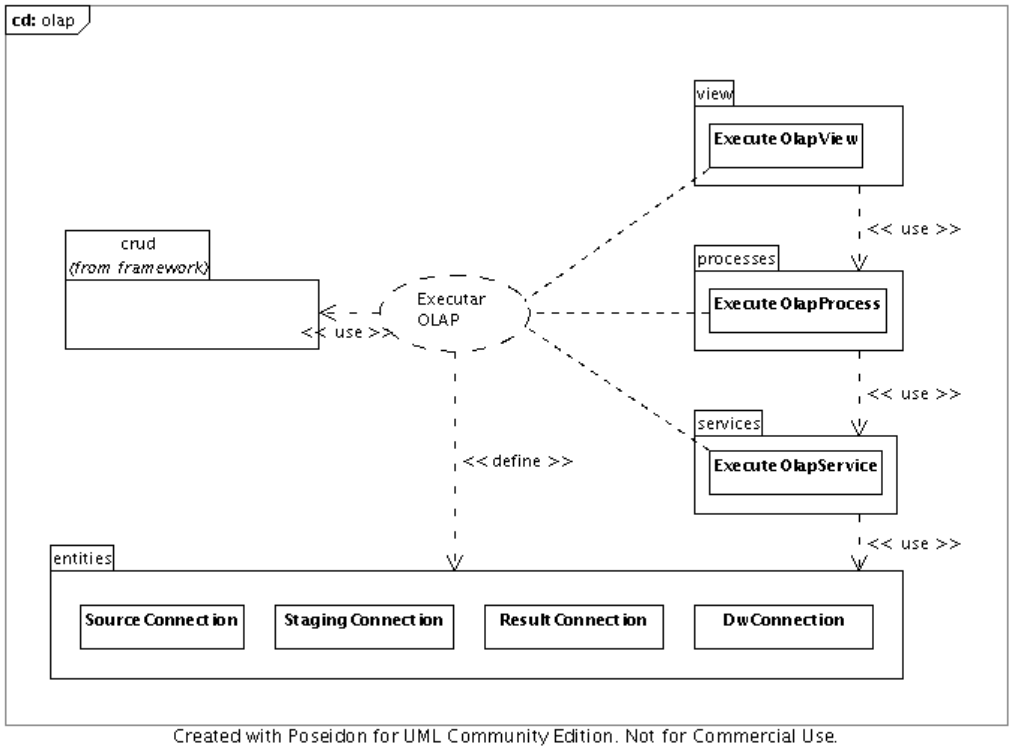


Figura 3.19: Serviço, processo, visualização e dependência do caso de uso *Executar OLAP*.

Executar DM

A execução de mineração de dados consiste, basicamente, em preparar um conjunto de dados, definir uma técnica de mineração, aplicar a técnica sobre os dados e analisar os resultados obtidos. Para a seleção do conjunto de dados, este trabalho utiliza os módulos de definição de repositórios de dados: *source*, *staging*, *dw* e *result* (mostrados na Figura 3.9). Por esta razão este caso de uso utiliza as entidades de conexão definidas pelos casos de uso *Definir Origem*, *Definir Estágio*, *Definir Dw* e *Armazenar Resultados*. Para a preparação dos dados são propostos os módulos *etDefinition* e *loadDefinition*. Além destes, o próprio módulo *olap*

serve como um preparador de dados que pode armazenar seus resultados no módulo *result*. Esses resultados podem ser utilizados para mineração. Com estas opções, busca-se tornar a arquitetura proposta bastante flexível quanto à preparação dos dados. Uma vez que estas atividades de preparação consomem a maior parte do tempo da equipe que desenvolve um sistema KDD.

Para a realização deste caso de uso, foram definidos um serviço *ExecuteDmService*, um processo *ExecuteDmProcess* e uma visualização *ExecuteDmView*. O módulo CRUD é utilizado para obter as entidades de conexão dos repositórios de dados, bem como visualizar todo o conteúdo das estruturas de cada conexão: tabelas e itens de dado por elas acessados.

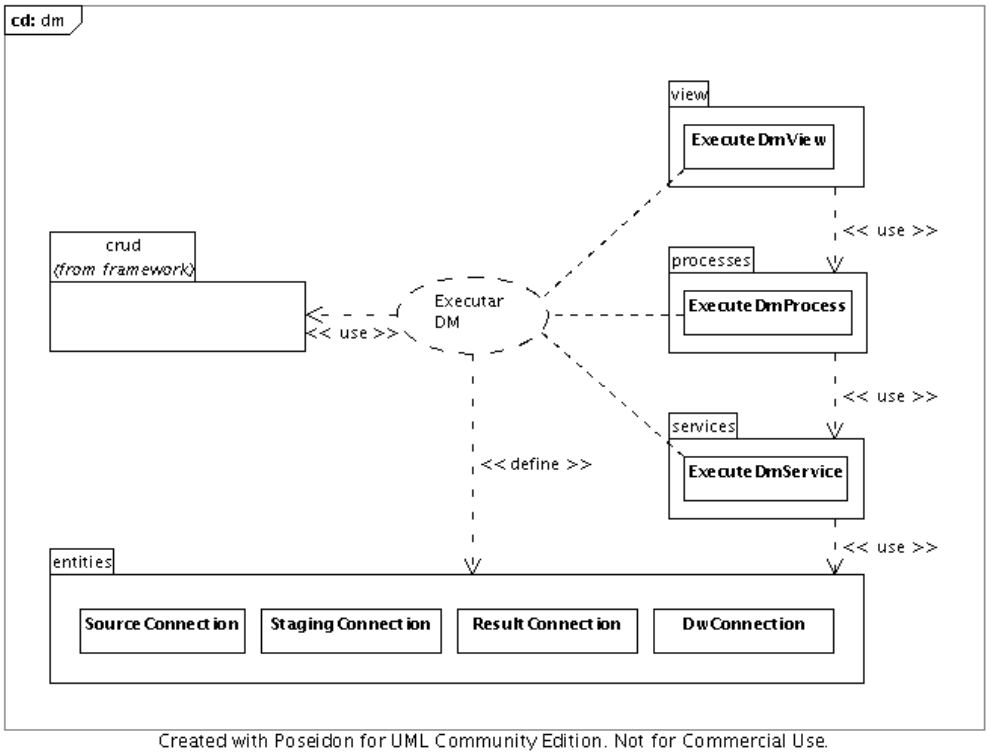


Figura 3.20: Serviço, processo, visualização e dependência do caso de uso *Executar DM*.

Visualizar resultados

A Figura 3.21 mostra os serviços, processo, visualização e dependência deste caso de uso. Foi definido um serviço *IntegrationService* que é responsável por prover a integração de ferramentas de visualização de resultados com o sistema. Caso a equipe escolha desenvolver algumas ferramentas de visualização de resultados, ela poderá utilizar as classes *ViewResultService*, *ViewResultProcess* e *ViewResultView* para implementar estas funcionalidades.

Esta parte de visualização de resultados ainda não foi bem explorada pela atual proposta. Espera-se que com o desenvolvimento de outros trabalhos específicos nesta área, alguns refinamentos sejam realizados de forma a tornar o componente de visualização da arquitetura tão versátil quanto os componentes de extração, transformação, *estagiamento* e carga, já definidos.

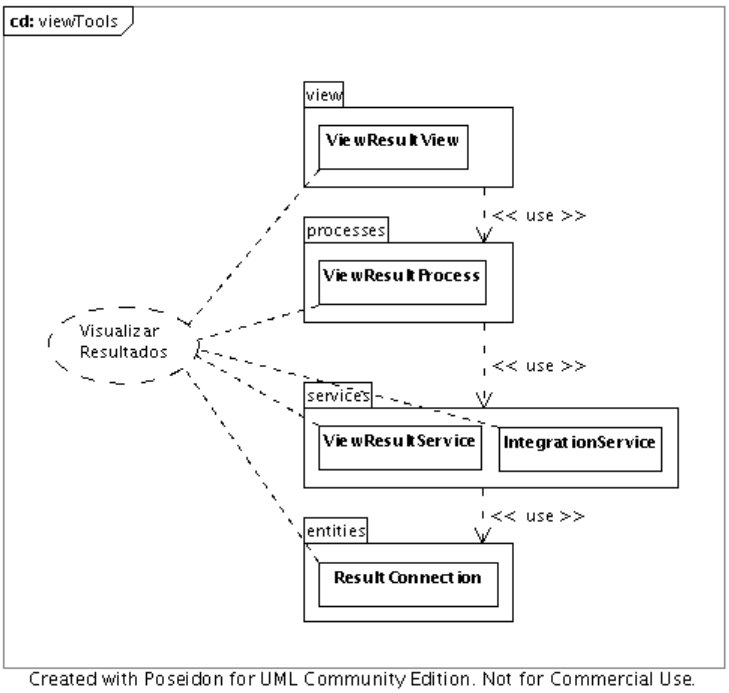


Figura 3.21: Serviços, processo, visualização e dependência do caso de uso *Visualizar Resultados*.

3.5. Comparação da arquitetura proposta com trabalhos relacionados

A arquitetura proposta buscou reunir as principais características dos atuais trabalhos relacionados a ambientes de descoberta de conhecimento em banco de dados.

A Tabela 3.8 apresenta uma comparação entre algumas características dos trabalhos relacionados e da arquitetura proposta. O principal diferencial deste trabalho é a união da maioria das características em uma única arquitetura que possibilite a construção de ambientes de descoberta de conhecimento corporativos.

Tabela 3.8: Comparação entre as características dos trabalhos relacionados e da arquitetura proposta

Características	Oracle (2005)	Gupta (2004)	GridMiner (2003)	KDB200 (2006)	Arquitetura proposta
Linguagem descritiva	✘	✔	✔	✘	✘
Atividades interativas	✘	✘	✘	✔	✔
Arquitetura de Software reutilizável	✘[1]	✘[2]	✘[3]	✘[4]	✔[5]
Metadados das fontes e destinos de dados e do processo	✔	✔	✘	✘	✔
Construção de DW	✔	✘	✔	✘	✔
Atividades OLAP	✘	✔	✔	✔	✔
Atividades DM	✘	✔	✔	✔	✔

3.6. Considerações finais

A arquitetura de referência guia a especificação da arquitetura de software do sistema KDD. Como foi visto em sua concepção, alguns problemas foram levantados e algumas soluções foram propostas. Tendo em vista a não especialização da arquitetura de referência, as soluções propostas não estão ligadas especificamente ao domínio KDD, são soluções mais genéricas, mantendo assim, o objetivo de uma arquitetura de referência. Problemas que exigem uma solução específica para o domínio KDD são resolvidos na arquitetura de software.

Com a utilização das visões da arquitetura, foram mostrados os aspectos estáticos e dinâmicos. Alguns detalhes de implementação que viriam a esclarecer algumas dúvidas sobre o funcionamento foram deixados para o próximo capítulo. Contudo, as visões mostram-se bastante explicativas e permitem apresentar tanto a organização mais geral da arquitetura quanto seu funcionamento mais interno.

A arquitetura de referência proposta mostra-se bastante independente do domínio KDD, tornando-a reutilizável para outros domínios. Especificamente, nota-se que os problemas por ela resolvidos são bastante comuns aos problemas do domínio de sistemas de informação.

---

[1] Arquitetura proprietária da Oracle.

[2] Definida somente para o ambiente do autor.

[3] Conj. de componentes reutilizáveis que deixam a arquitetura do sistema KDD por conta da equipe que utiliza os componentes.

[4] Definida somente para a aplicação KDB2000 do autor.

[5] Definida para que equipes de desenvolvimento possam estendê-la. Com separação da arquitetura de referência e da arquitetura de software.

Questões de portabilidade de plataforma, de dados e de interface são básicas para qualquer software que pretende se integrar a ambientes corporativos. Um ponto que poderia ser discutido com mais detalhe é a especificação de gerenciadores de processos e serviços para ambientes distribuídos. Esta característica contribuiria para a utilização desta arquitetura de referência em sistemas mais críticos. Porém, esta é uma sugestão para trabalhos futuros, devido ao escopo e cronograma do atual trabalho.

O módulo CRUD proposto nesta arquitetura se mostra um mecanismo avançado para controlar as estruturas internas do sistema KDD. Essas estruturas são os metadados do sistema. Elas descrevem a organização do processo KDD, indicam quais conexões são usadas, quantas tabelas e itens de dado estão disponíveis, quais funções de transformação e carga dos dados que são utilizados. Estes seriam os metadados técnicos definidos por David (2001).

Como pôde ser visto, a definição dos componentes da arquitetura de software foi baseada nos conceitos definidos na arquitetura de referência, tendo como foco principal o domínio da aplicação. As questões relacionadas à comunicação entre componentes, segurança e outras ficaram na definição da arquitetura de referência, possibilitando, assim, uma melhor definição do domínio da aplicação e das funcionalidades do sistema na arquitetura de software.

Neste trabalho foram descritos detalhadamente os componentes de extração, transformação e carga dos dados, de forma a mostrar as estruturas que controlam e tornam mais automáticas estas atividades. Além disso, abstração de serviços, processos e visualizações facilita a implementação de inúmeras funcionalidades de apoio ao processo KDD.

A proposta da arquitetura de software deste trabalho não restringe a especificação das funcionalidades do módulo às atuais estruturas descritas pelos casos de uso. Elas servem somente como um guia inicial para solucionar problemas mais simples. Como também, para a construção de um protótipo mais simplificado para validação da arquitetura proposta.

Finda aqui o capítulo que descreve e documenta a arquitetura para sistemas KDD proposta. O próximo capítulo descreve o *framework* que foi implementado neste trabalho. Este *framework* auxiliou na validação da arquitetura, permitindo identificar conceitos que no momento da implementação mostraram-se inadequados e foram alterados. Após a descrição do *framework* é apresentado o protótipo e um estudo de caso para validação desta proposta.

4. – O *Framework*

Durante o desenvolvimento deste trabalho houve uma preocupação em pesquisar não somente sobre a definição da arquitetura, mas também sobre problemas relacionados ao desenvolvimento de software centrado em arquitetura. Isto porque, muitas vezes, uma arquitetura representada por diagramas ou por uma ADL (*Architecture Definition Language*) pode até ser bem compreendida pelo analista responsável, porém haverá muitas dúvidas sobre como implementar a arquitetura, ou seja, sobre como transformar os diagramas construídos (casos de uso, classes, pacotes, etc.) em linhas de código.

Para facilitar a utilização da arquitetura de software proposta foi desenvolvido um *framework* que implementa os componentes básicos desta arquitetura. Este *framework* é como o esqueleto dorsal de todo o software, ele implementa os componentes que podem ser estendidos para integração das funcionalidades do software.

Além dos componentes do *framework*, outra colaboração deste trabalho é a configuração do ambiente de desenvolvimento utilizando a tecnologia Java, incluindo integração dos trabalhos da equipe em um repositório central e a organização dos projetos, entre outras questões.

Primeiramente, na seção a seguir são apresentadas as questões de desenvolvimento de software que foram tratadas durante este trabalho. Logo depois, iniciam as seções que descrevem as partes do *framework*.

4.1. Questões de desenvolvimento de software

Como este trabalho propõe uma arquitetura, houve sempre a preocupação de verificar como cada solução proposta seria implementada; como cada diagrama poderia ser transformado em pacotes, classes e linhas de código. No entanto, para a transformação de diagramas em códigos existem algumas questões que o autor deste trabalho decidiu levantar antes do início da implementação.

A primeira questão diz respeito à linguagem que seria utilizada para a implementação. Como um sistema KDD é integrado em ambientes corporativos[1], a primeira decisão foi de escolher uma tecnologia que ofereça suporte ao desenvolvimento de sistemas corporativos. Foram analisadas as plataformas *.Net*[2] da Microsoft e a *J2EE* da Sun. A segunda foi escolhida por ser um software gratuito e manter um bom relacionamento com a comunidade de software livre[3], além de contar com inúmeras ferramentas gratuitas.

Além da questão da linguagem, que acabou envolvendo toda uma plataforma de desenvolvimento, outras questões precisaram ser abordadas. Os itens a seguir apresentam as questões e a soluções encontradas.

4.1.1. Melhores práticas em desenvolvimento J2EE

Broemmer (2003) apresenta práticas de desenvolvimento que durante anos têm sido comprovadamente as melhores. Seu foco é na plataforma *J2EE*, no entanto, as questões abordadas são perfeitamente praticadas em qualquer plataforma ou metodologia de desenvolvimento de software. Para este trabalho foram separadas três práticas que desempenharam um importante papel na especificação do *framework* desenvolvido.

Tratamento padrozinado das mensagens geradas pelos elementos da arquitetura

Durante a execução de uma atividade, alguns erros podem acontecer. O usuário que iniciou a atividade precisa ser informado sobre o que deu errado. A exibição de uma pilha de execução que mostra os procedimentos que foram interrompidos não será esclarecedora para o usuário. É necessário que o sistema informe, de uma maneira sistemática, quais atividades de negócio foram interrompidas e como o usuário pode proceder. O recurso de tratamento de exceções das linguagens é um avançado mecanismo que auxilia neste controle de erros e de mensagens.

---

[1] Ambiente corporativo é referido aqui ao ambiente de instituições que utilizam diversas soluções de software e que integram seus processos de negócio com os seus sistemas de informação. Muitas vezes este ambiente é heterogêneo, com a presença de diversas tecnologias.

[2] Maiores informações sobre a plataforma .NET da Microsoft podem ser conseguidas no sítio http://microsoft.com/net.

[3] Uma referência de software livre no país é o portal do próprio governo (http://www.softwarelivre.gov.br). Apesar do Java não ser um software livre, a Sun, sua proprietária, apóia as iniciativas de software livre (http://www.softwarelivre.gov.br).

No entanto, ele é bastante técnico e é focado em tratar exceções das rotinas do software. É necessário estendê-lo para criar um mecanismo capaz de controlar exceções de negócio.

Erros de variáveis não inicializadas, de tipos incompatíveis de dados, de índice inválido de vetor, entre outros, são erros da linguagem que devem ser separados dos erros de negócio, que seriam: erro ao iniciar um processo; erro ao executar um serviço; erro ao registrar a movimentação na conta; erro de saldo insuficiente para a operação. Geralmente, erros de linguagem revelam *bugs* da aplicação, enquanto que erros de negócio revelam inconsistência nos dados da aplicação ou dos parâmetros fornecidos para algum processo.

Estendendo o mecanismo de exceções, foi criada uma estrutura de mensagens para que cada serviço, processo, visualização ou componente de apoio possa tratar as mensagens de negócio de uma maneira padronizada. Os erros de linguagem são convertidos para esta estrutura. Este mecanismo também é utilizado para tratar mensagens de informações que devem ser exibidas para o usuário. Os detalhes deste mecanismo são apresentados na seção *4.2.5-Exceções e Mensagens*.

Mensagens armazenadas fora do código, em repositório de mensagens

Os textos das mensagens são mais voláteis que o código da aplicação, durante o período de implantação as mensagens tendem a ser alteradas para serem mais bem compreendidas pelo usuário. Uma boa prática é armazenar estas mensagens fora do código fonte da aplicação. Isto permite que os textos sejam alterados sem que a aplicação seja compilada novamente. Além disso, os erros de negócio mostrados para o usuário devem estar na linguagem utilizada pelo usuário. Isto implica que o mecanismo de mensagem precisa armazenar os textos das mensagens em diversas linguagens, o que contribui para a utilização de um repositório que armazena as mensagens em diversas linguagens.

O mecanismo de mensagem implementado neste trabalho segue estas especificações.

Metadados sobre as entidades em arquivos separados

É comum que os programadores implementem rotinas de validações das propriedades das entidades dentro do código da aplicação como por exemplo: valor máximo, mínimo, valores válidos, entre outras. No entanto, algumas validações podem ser automatizadas utilizando um repositório de metadados que descrevem as validações que devem ser aplicadas às

propriedades das entidades. Isto permite que as validações sejam alteradas sem alterar linhas de código da aplicação. Além disso, centraliza a definição das validações utilizadas pela aplicação, evitando a redundância de código e facilitando a manutenção do software.

Seguindo estas especificações, foi implementado o módulo CRUD, que se mostrou um mecanismo avançado para manipulação de entidades e dos metadados destas entidades. A Seção *4.3.3*-(CRUD) apresenta os detalhes de implementação deste módulo.

4.1.2. Problema de controle de instâncias de objetos e relacionamento entre os objetos criados

Uma arquitetura deve manter um controle sobre os objetos por ela instanciados de forma que ela possa definir o relacionamento entre os objetos de uma maneira mais automática. Além disso, é necessário que muitos objetos criados pela arquitetura recebam referências da própria arquitetura em que ele está sendo instanciado. Esta questão é resolvida pelo padrão de projeto chamado fábrica de objetos (Metsker, 2002). Para a implementação do *framework* foi utilizado um outro *framework* de integração chamado *Spring*[1].

O *Spring* é um *framework* de código aberto que foi desenvolvido com o objetivo principal de facilitar o desenvolvimento de aplicações empresariais. Sua principal característica é uma fábrica de instâncias de objetos. Esta fábrica possui um mecanismo de IoC (*Inversion of Control*) que, ao instanciar um novo objeto, verifica os relacionamentos do objeto, instancia outros objetos necessários e faz a ligação entre os objetos. Isto evita que o programador fique. controlando os objetos instanciados e os relacionamentos entre eles.

É comum o desenvolvedor de software querer desenvolver seu próprio *framework* de controle de instâncias de objetos. No entanto, a utilização de um *framework* como o *Spring* pode oferecer inúmeros outros benefícios para o projeto, como por exemplo:

- Possibilidade de integração de outras ferramentas como o Ant[2], JSP[3] , Hibernate[1] e xDoclet[2]. Isto seria, no mínimo, bastante trabalhoso de se obter com uma fábrica personalizada de objetos;

[1] http://www.springframework.org

[2] http://ant.apache.org

[3] http://java.sun.com/products/jsp

- Mecanismo de Programação Orientada a Aspectos (AOP[3], do inglês *Aspect Oriented Programming*): Por ser uma fábrica de objetos, o *Spring* tem total controle sobre o objeto instanciado, com isto, é possível definir a injeção de aspectos em tempo de execução da aplicação utilizando a configuração da fábrica;
- Arquivo de configuração centralizado: O *Spring* utiliza um arquivo XML que descreve a aplicação que ele vai gerenciar. Neste arquivo são definidas as classes, os métodos de instanciação e os relacionamentos entre os objetos instanciados. Vários parâmetros da aplicação podem ser alterados simplesmente editando este arquivo, sem a necessidade de re-compilar a aplicação;

4.1.3. Problema de persistência de objetos utilizados na arquitetura em banco de dados relacional

A arquitetura de referência define algumas entidades que são persistidas e recuperadas para utilização durante a execução do sistema KDD. Estas entidades na verdade são objetos que devem ser armazenados em algum repositório de objetos. Como o sistema KDD geralmente é instalado no ambiente computacional de uma instituição junto com diversas outras ferramentas, foi definido que o *framework* não deve obrigar a instalação de um gerenciador de banco de dados orientado a objetos, mas que utilize os atuais gerenciadores de banco de dados relacionais. Como o problema de armazenamento de objetos em estruturas relacionais já foi bastante pesquisado e apresenta algumas soluções satisfatórias, foi utilizado o *framework Hibernate* para resolver esta questão na arquitetura.

O *Hibernate* é um *framework* de persistência de objetos sobre bancos de dados relacionais de maneira transparente. É considerado um dos maiores projetos de código aberto desenvolvido em Java. As principais vantagens da utilização do *Hibernate* em um projeto são (Bauer e King, 2005):

- Transparência do mapeamento OO vs. Relacional: Dentro da aplicação são vistos objetos e coleções de objetos sem a preocupação de referência de esquemas, tabelas e itens de dado do banco de dados. Isto permite que o programador concentre seus

---

[1] http://www.hibernate.org

[2] http://xdoclet.sourceforge.net

[3] http://aosd.net

esforços na aplicação dos conceitos OO nas entidades e nos relacionamentos desta entidade. Utilizando o *Hibernate* para o controle de persistência, o programador não necessita implementar classes que realizam as operações de inserção, alteração e remoção de uma determinada entidade de negócio.

- Portabilidade de banco de dados: Utilizando o *Hibernate*, todas as classes da aplicação são mapeadas pelo *framework*. Este mapeamento é independente do banco de dados a ser utilizado e é responsabilidade do *Hibernate* realizar as adaptações e traduções do mapeamento para instruções SQL compatíveis a cada sistema gerenciador de banco de dados disponível no mercado.
- Linguagem de consulta de objetos: Outra característica muito importante do *Hibernate* é fornecer uma linguagem de consulta bastante parecida com a SQL, a HQL (*Hibernate Query Language*). A HQL permite realizar consulta de objetos persistidos utilizando os conceitos OO. Esta linguagem é bastante flexível e suas consultas apresentam um grau de compreensão maior do que a mesma consulta escrita em SQL. Isto porque as relações entre as classes de objetos ficam transparentes para quem escreve a consulta. Por ser bastante semelhante à SQL, a HQL é de fácil aprendizado.

4.1.4. Problema *de documentação*

A questão aqui tratada é a documentação das interfaces e códigos desenvolvidos. Fazendo a pergunta: quem é que gosta de documentar o que implementa, em uma sala de aula de bacharelandos em informática ou ciência da computação é possível notar que a documentação do software pode se tornar um problema se não abordada logo no início do projeto.

Durante o desenvolvimento de uma rotina, a atenção do programador está voltada à resolução do problema. A documentação geralmente é deixada para um segundo momento, que às vezes não chega nunca. Para auxiliar nesta questão, a integração da documentação com o próprio código é uma proposta que evita que o programador tenha que acessar outra ferramenta para documentar o que está sendo implementado. Segundo Pamplona (2006), a linguagem Java inventou o conceito de comentário de documentação. Este comentário é específico para quem

precisa saber o que o código fonte faz sem ver o código, ou seja, é um comentário para documentos. Este padrão de documentação é chamado de *JavaDoc*[1].

4.1.5. Problema de testes

A Figura 4.1 mostra o impacto das alterações no decorrer do ciclo de vida de um projeto de software. Murphy (2005) destaca a importância de estar definindo testes logo no início do processo de desenvolvimento de um software. Ele mostra que é indispensável que cada funcionalidade do sistema seja testada antes de sua integração com os demais elementos da aplicação. Apesar deste autor propor um desenvolvimento dirigido por casos de teste, que não foi o paradigma utilizado por este trabalho, as questões de testes abordadas contribuíram para a implementação de classes que auxiliam na realização de testes em funcionalidades que se integrarão à arquitetura.

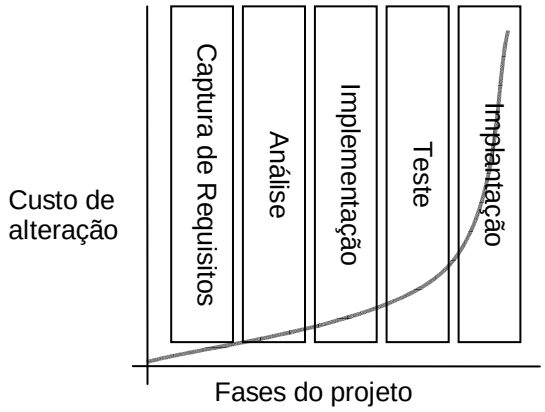


Figura 4.1: Relação custo de alteração vs. fase do projeto. Adaptado de (Murphy, 2005).

Para a implementação das classes básicas para teste foi utilizado o *framework* de teste unitário JUnit[2]. Com este *framework,* é possível construir classes de testes que são instanciadas e executadas para automatizar as atividades de testes.

4.1.6. Problema de integração do desenvolvimento da equipe

O desenvolvimento de um sistema de descoberta de conhecimento envolve inúmeros profissionais como gerentes de negócio, especialista em banco de dados, engenheiros de

[1] http://java.sun.com/j2se/javadoc

[2] http://www.junit.org

software e programadores. Trata-se de um sistema de grande porte, mas, no entanto, um sistema KDD possui partes bem definidas que podem ser desenvolvidas por diferentes equipes. Para integrar os trabalhos realizados é necessário o uso de uma ferramenta de controle de versão concorrente (CVS, do inglês *Concurrent Version System*).

O ambiente escolhido para o desenvolvimento do *framework* foi o Eclipse por possuir uma interface ágil, inúmeros recursos que facilitam a produção de software (assistentes e modelos) e consumir menos recurso computacional do equipamento (é mais leve). Neste ambiente, o sistema de controle de versão já é integrado. Sendo necessário somente configurar um servidor do repositório central.

4.1.7. Problema de composição de arquivos de configuração da aplicação

Por se tratar de um ambiente de desenvolvimento que envolve vários *frameworks*, a atividade de definir os arquivos de configuração destes *frameworks*, para manipular os componentes da arquitetura pode se tornar uma tarefa bastante extensa. Esses *frameworks* são configurados por meio de arquivos XML. As alterações de uma funcionalidade ou a criação de uma nova funcionalidade exigem que os arquivos de configuração dos *frameworks* sejam atualizados. Para agilizar estas alterações foi utilizada a ferramenta *XDoclet*[1].

XDoclet é um *framework* que possibilita programação orientada a atributos. Utilizando os comentários de documentação *JavaDoc* é possível adicionar metadados no código fonte Java. A ferramenta analisa gramaticalmente o código fonte e gera os metadados em arquivos XML. Esses arquivos são utilizados como arquivo de configuração por outras ferramentas como *Spring*, *Hibernate* e *Java Faces*. Desta forma, a manutenção das configurações é facilitada por que tudo se encontra dentro de um mesmo arquivo, o código fonte.

Para acionar as tarefas da ferramenta *XDoclet* é utilizada uma ferramenta de compilação chamada *Ant*[2]. Esta ferramenta permite a criação de arquivos de configuração XML que descrevem todo o processo de compilação e implantação de uma aplicação. Seu

[1] http://xdoclet.sourceforge.net

[2] http://ant.apache.org

principal objetivo é substituir os complicados comandos de console que são necessários para compilar, configurar, copiar e integrar uma aplicação. Estas atividades são abstraídas em tarefas que podem ser facilmente configuradas e executadas.

4.1.8. Juntando tudo

Todas estas questões respondidas formaram uma base sólida para a implementação de um *framework* que vai além de um simples protótipo para a validação da arquitetura. As funcionalidades e a estabilidade do *framework* implementado o torna uma importante ferramenta para a continuação deste projeto e implementação de outros projetos que necessitem de uma arquitetura de referência e um *framework* já implementado.

A Figura 4.2 mostra a área de trabalho do ambiente de desenvolvimento do *framework* e a integração das ferramentas em um único ambiente. A seguir é descrito cada item que foi destacado na figura.

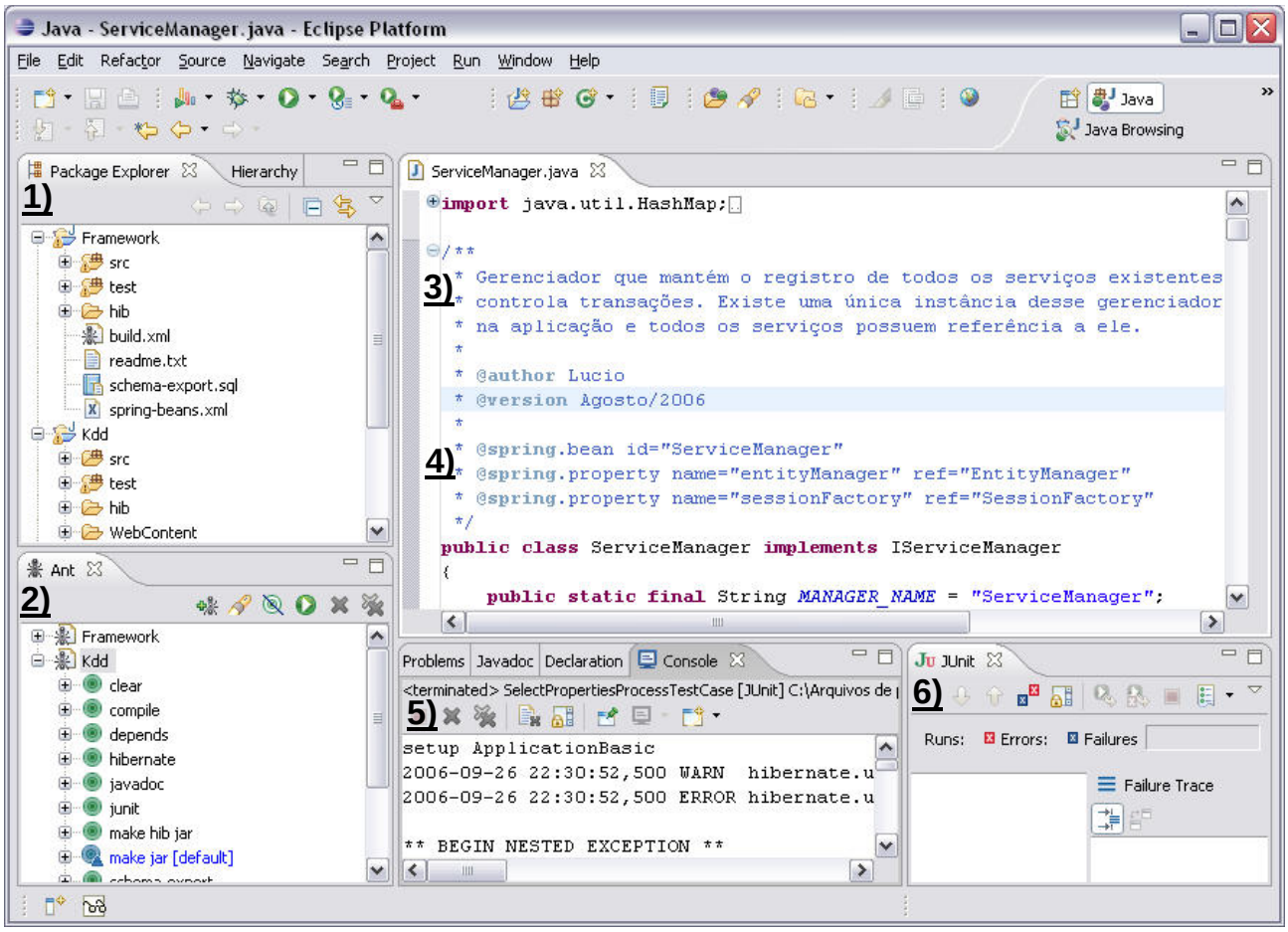


Figura 4.2: Área de trabalho no ambiente de desenvolvimento Eclipse .

O item 1 da figura destaca a estrutura dos projetos *framework* e *Kdd*. Como pode ser visto, cada projeto possui especificações separadas e é implementado independentemente. Porém, todos seguem uma estrutura básica de diretórios que foi definida para melhor organizar os artefatos dos projetos. Na Tabela 4.1 são descritos os diretórios e os arquivos em destaque no item 1.

Tabela 4.1: Descrição dos diretórios e arquivos do projeto de implementação do *framework*.

Diretório	Descrição
src	Diretório onde os pacotes e as classes Java são armazenados.
test	Diretório onde os pacotes e as classes que implementam os testes unitários são armazenados. Nesta pasta é seguida a mesma hierarquia de pacotes da pasta *src*.
bin	Este diretório não aparece na figura por ser um diretório oculto, porém ele é utilizado pelo compilador para armazenar as classes compiladas.
hib	Este diretório armazena os arquivos de configuração do *Hibernate* e do *Spring.*
WebContent	Este diretório segue a estrutura J2EE que permite a implantação de uma aplicação em diversos servidores de aplicação que seguem o padrão J2EE. A estrutura interna deste diretório é mostrada na Figura 3.5 da Seção *3.3.2*-Documentação deste trabalho.
Arquivo	Descrição
build.xml	Este é o arquivo de configuração da ferramenta de compilação *Ant*. O item 2 da figura mostra como este arquivo é organizado em tarefas. Cada tarefa é configurada dentro do arquivo e pode ser acionada com um duplo clique sobre o ícone da mesma.
readme.txt	Este arquivo descreve a estrutura de diretório definida.
schema-export.sql	Este arquivo é gerado pelo *Hibernate* e contém as instruções SQL que criam as tabelas, itens de dado, índices e restrições relacionais em um banco de dados. Este arquivo é criado de acordo com o mapeamento dos objetos persistidos que são manipulados pelo sistema.
spring-beans.xml	Este arquivo é utilizado pelo *Spring* para definir algumas configurações adicionais da aplicação.

O item 3 da figura destaca a documentação integrada ao código que posteriormente é analisada pela ferramenta *JavaDoc* para geração de documentos.

O item 4 da figura destaca a inserção de metadados que são utilizados pela ferramenta *XDoclet* para geração dos arquivos XML de configuração. Neste item, são mostrados metadados que definem a atual classe *ServiceManager* como uma unidade controlada pela ferramenta *Spring*.

O item 5 da figura destaca algumas visualizações do ambiente. Em evidência está a visualização do *Console* onde são mostradas as mensagens de execução da aplicação, entre outras funcionalidades.

O item 6 da figura destaca a ferramenta de teste unitário *JUnit*. Esta ferramenta executa os testes e indica quais foram executados com sucesso ou com falha.

Com isto, o ambiente de desenvolvimento está montado. Nas próximas seções é apresentada a descrição detalhada do *framework*.

4.2. Núcleo do framework

O *framework* está dividido em quatro grandes módulos: o módulo que implementa o seu núcleo, o módulo de segurança, o módulo de auditoria e o módulo CRUD. Nesta seção são apresentados os componentes do núcleo.

O núcleo implementa as funcionalidades mais básicas do *framework*. Nele é definido todo o padrão de instanciação e comunicação entre os componentes da arquitetura. Ele define as interfaces dos componentes e implementa os gerenciadores de cada camada. Na arquitetura de referência foram especificadas as responsabilidades e características de cada módulo. A implementação do *framework* segue rigorosamente estas especificações.

Nos itens a seguir, são apresentados detalhes de implementação das definições da arquitetura.

4.2.1. Os pacotes do núcleo

A Figura 4.3 mostra os pacotes que foram definidos para organizar as funcionalidades do núcleo da arquitetura. Nestes pacotes são incluídas as interfaces e implementações dos gerenciadores que compõem o esqueleto dorsal da arquitetura. Cada camada da arquitetura é representada por um pacote (*view*, *process* e *service*). Todo componente que pretende integrar-se à arquitetura fará uma referência a algum elemento destes pacotes. Além dos pacotes que definem os componentes da arquitetura, também são definidos como parte do núcleo da arquitetura os pacotes de apoio aos testes, tratamento das exceções e utilitários (*test*, *exception* e *util*).

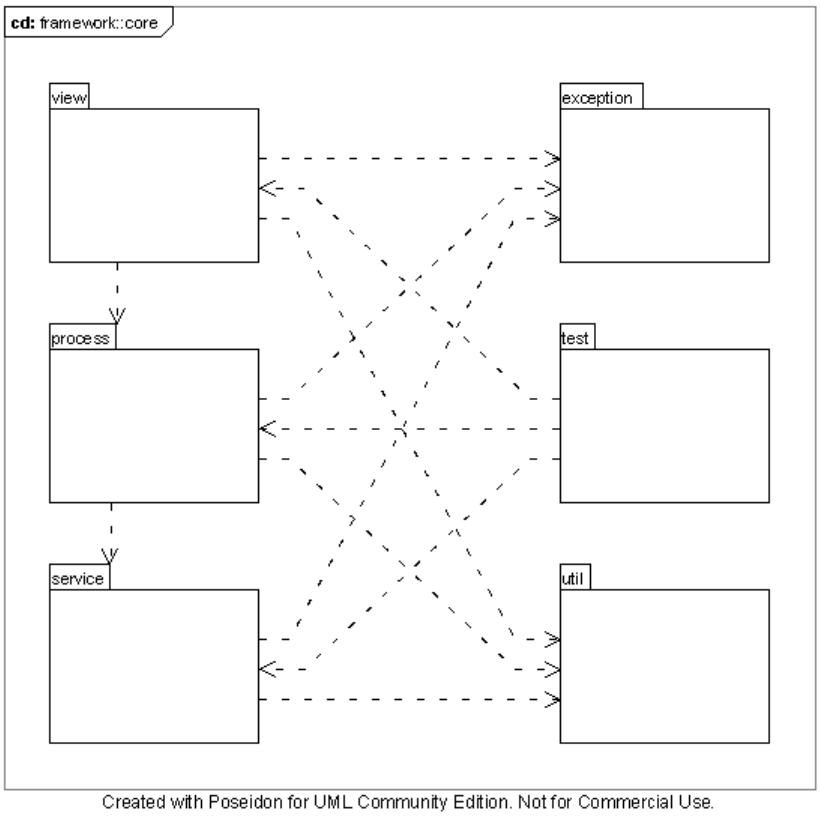


Figura 4.3: Pacotes que compõem o núcleo do *framework*.

4.2.2. O pacote de serviço

Seguindo as definições do estilo arquitetural orientado a serviços (SOA), o conceito de serviço especifica uma rotina de software que é invocada com alguns parâmetros e possui dois fluxos de execução: sucesso ou fracasso. No primeiro, o serviço retornará ao invocador o resultado do processamento dos parâmetros de entrada e no segundo, o serviço indicará que houve algum problema durante a sua execução e que não foi possível concluir suas atividades. Durante a execução de um serviço não é possível interagir com o mesmo. Nestes aspectos, um serviço é semelhante a uma função, porém, como será mostrado logo a seguir, o serviço é um componente mais sofisticado.

Para a invocação de serviços foi definido um gerenciador que é responsável pela verificação da disponibilidade do serviço, sua localização e execução. Este gerenciador exige o preenchimento de uma estrutura que descreve o serviço a ser invocado e os parâmetros de execução. Com esta estrutura, o gerenciador identifica o serviço, inicia e monitora sua execução.

As principais características deste gerenciador são:

- Padronização na invocação de um serviço: todo serviço segue uma interface bem definida que permite ao gerenciador de serviços controlar os serviços disponíveis na arquitetura. Esta interface possui três atividades básicas: 1) registrar-se no gerenciador quando o serviço é instanciado pela arquitetura; 2) receber do gerenciador a estrutura com os parâmetros de execução; 3) retornar para o gerenciador o resultado de sua execução ou uma indicação de falha. Esta interface simplifica a implementação e a integração de novos serviços ao sistema.

- Monitoramento dos serviços em execução: com um controle centralizado de execução de serviços, é possível monitorar todas as execuções que estão sendo realizadas dentro do sistema. Com isto, é possível controlar o tempo de execução de cada serviço e gerar informações sobre os serviços mais executados. Apesar da atividade de monitoramento não aparecer freqüentemente nas primeiras definições de um sistema, nota-se que, com o crescimento do sistema, a atividade de monitoramento torna-se um importante guia para a identificação de gargalos ou pontos que estejam comprometendo o desempenho. Na atual implementação do framework, o gerenciador não pode interromper a execução de um serviço, somente iniciar sua execução e aguardar seu encerramento.

- Controle de serviços transacionais: antes da execução de um serviço, o gerenciador identifica se o serviço deve ser executado dentro de uma transação, cria uma transação e monitora a execução do serviço para decidir se a transação será confirmada (*commit*) ou desfeita (*rollback*). Esta definição simplifica a implementação dos serviços, pois os mesmos não precisam conter, dentro do seu código de execução, rotinas para controle de transação. Todo este controle fica a cargo do gerenciador. No *framework* implementado, uma propriedade do serviço indica se ele deve ser tratado como um serviço transacional ou não.

Como citado anteriormente, em conjunto com o gerenciador é utilizada uma estrutura que fornece todas as informações necessárias para a execução de um serviço. Esta estrutura é preenchida com o nome do serviço que será invocado e os parâmetros de execução. A estrutura que informa os parâmetros de execução do serviço é representada pela classe *ServiceData* na Figura 4.4. Esta classe possui uma lista de parâmetros que devem ser preenchidos de acordo com o serviço que será executado. Estes argumentos são compostos de um nome, que identifica o parâmetro do serviço, e de um valor. Esta técnica de fornecimento

de parâmetros por nome e valor torna a integração dos serviços mais simples, pois possibilita que sejam fornecidos inúmeros parâmetros para um serviço, sem ter uma referência à atual instância deste serviço. Outra vantagem é que uma descrição textual com o nome do serviço, o nome de alguns parâmetros e os valores destes parâmetros, pode ser automaticamente convertida para a estrutura *ServiceData* e enviada para um gerenciador de serviços executá-la. Desta forma, tem-se um mecanismo bastante flexível para invocação de serviços. Vale citar também que o desacoplamento entre os serviços disponíveis e o restante da arquitetura contribui para a manutenibilidade do software e, conseqüentemente, para sua qualidade. Esta estrutura é uma adaptação da estrutura apresentada por Broemmer (2003).

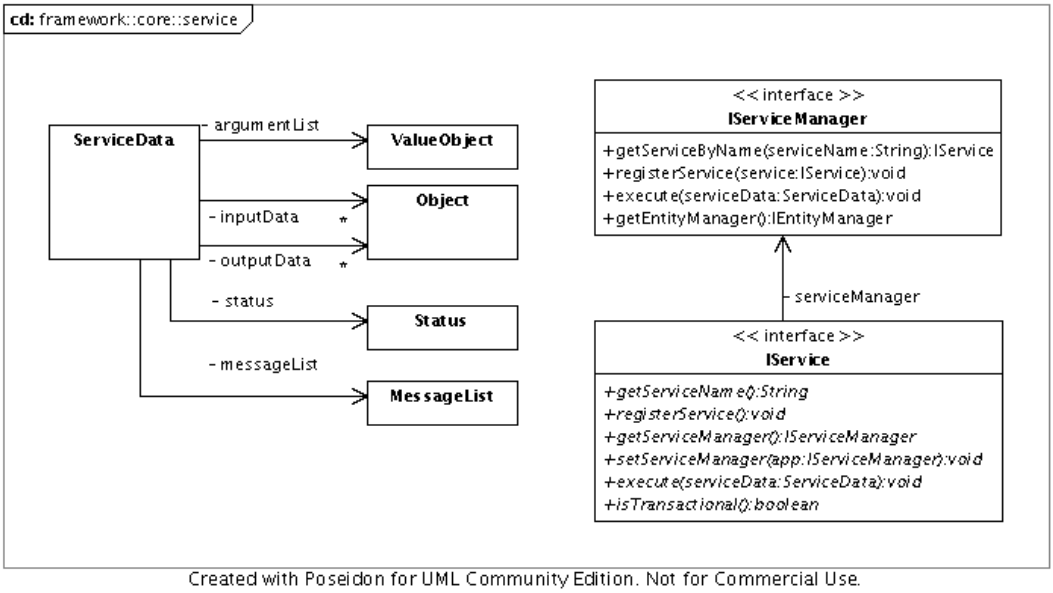


Figura 4.4: Classes que compõem o gerenciador de serviços do *framework*.

4.2.3. O pacote de Processo

Um processo já é uma rotina que realiza alguma atividade mais complexa. Esta atividade geralmente possui uma seqüência de passos definidos e diversos dados de entrada. A cada passo, o processo pode acessar serviços para obter alguma informação ou verificar se as informações fornecidas estão corretas.

A interface de um processo identifica os parâmetros suportados e os métodos das atividades que ele realiza. Todo processo deve estender a classe básica *ProcessBasic*. Esta classe implementa as rotinas de integração de um processo com o *framework* livrando o programador desta responsabilidade.

A Figura 4.5 apresenta as interfaces e as classes implementadas para o controle de processos. Vale lembrar que, pela definição da arquitetura de referência, o gerenciador de processo é autenticado e só permite que usuários autorizados acessem os processos. Outra característica do gerenciador é que mantém uma instância de processo para cada usuário, diferentemente do gerenciador de serviço que utiliza uma única instância de serviço para todos os usuários.

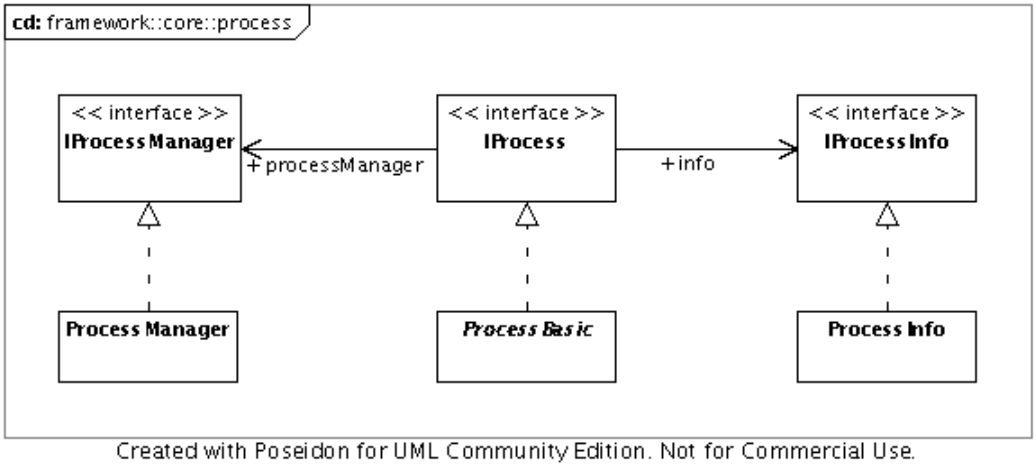


Figura 4.5: Interfaces que compõem a camada de processos da arquitetura de software.

4.2.4. O pacote de Teste

A realização de testes durante todo o desenvolvimento de um software é uma prática comprovadamente relacionada à qualidade do software. Seguindo esta prática, o *framework* proposto define um pacote com classes que auxiliam a equipe de desenvolvimento a realizar os testes dos componentes que são integrados na arquitetura.

A Figura 4.6 mostra as classes básicas para teste e uma classe de utilitários que também auxilia na realização dos testes. Dependendo da camada em que um determinado componente se encontra, durante seus testes ele necessitará de alguns recursos específicos da camada. Com isto, para cada camada foi implementada uma classe básica de teste que fornece ao componente testado acesso a toda a arquitetura. Por exemplo, para se testar um serviço, será necessário ter uma referência do atual gerenciador de serviços para poder invocar o serviço. Desta forma, a classe de teste do serviço deverá ser uma especialização da classe *ServiceBasicTest*. Da mesma forma, as classes de testes de processos deverão especializar a classe *ProcessBasicTest*, herdando assim os acessos à camada de processos e podendo acessar todos componentes das camadas inferiores.

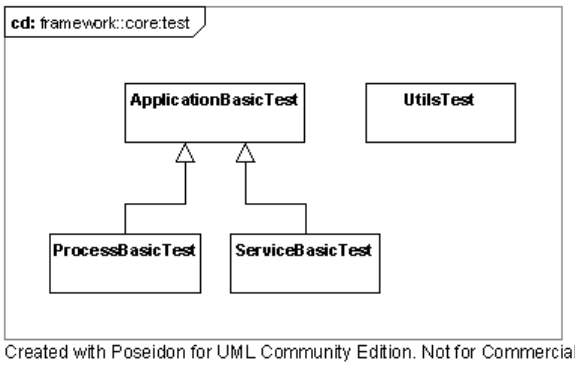


Figura 4.6: Classes básicas para realização dos testes unitários.

Para padronizar a implementação dos testes e possibilitar a utilização de ferramentas na automatização dos testes, foram definidas algumas convenções de nomenclatura dos pacotes, classes e métodos das classes de testes, que são:

- O nome da classe de teste deve receber o sufixo *TestCase*;
- O método de teste que será executado deve-se iniciar com o prefixo *test*;
- As classes de testes deverão ser organizadas em uma pasta separada das demais classes, dividindo assim, as classes que serão entregues para o ambiente de execução das classes que fazem parte dos testes;
- Para facilitar a localização dos testes, a mesma hierarquia de pacotes definida nas classes do projeto deverá ser seguida para as classes de testes.

A classe *ApplicationBasicTest* é a classe mais básica do pacote de teste. Ela implementa as funcionalidades para iniciar a aplicação.

A classe *ProcessBasicTest* prepara uma referência para o atual gerenciador de processos da aplicação. Assim, as classes de teste de processos podem utilizar esta referência já preparada para obter e testar um determinado processo.

A classe *ServiceBasicTest* prepara uma referência para o atual gerenciador de serviços da aplicação. Assim, as classes de teste de serviços podem utilizar esta referência já preparada para obter e testar um determinado serviço. Além disso, uma classe de teste que especializa esta classe terá acesso a todos os componentes que integram a camada de serviços e não terá acesso às camadas superiores.

A classe *UtilsTest* é estática e possui alguns métodos para controlar as estruturas de segurança da arquitetura e permitir a autenticação de operadores para a realização de testes de processos.

4.2.5. Exceções e Mensagens

Seguindo o princípio de padronização do tratamento de mensagens de informações e exceções, discutido por Broemmer (2003), a arquitetura define uma estrutura que controla a composição de mensagens. Além disso, segue a boa prática de separar mensagens de texto do código da aplicação. As mensagens são armazenadas em arquivos com extensão *.properties* que são identificados pelo nome da classe que pode gerar as mensagens contidas no arquivo. Isto possibilita que os arquivos *.properties* sejam escritos em diversas linguagens para serem utilizados.

O mecanismo de mensagem é composto de uma classe chamada *BusinessMessage*. Esta classe possui quatro propriedades básicas:

1. Tipo da mensagem: que indica se a mensagem é um erro ou uma informação. O tipo da mensagem é utilizado pelo gerenciador para identificar se a execução de um serviço foi concluída com sucesso. Se o gerenciador detectar que o serviço gerou alguma mensagem do tipo erro, o mesmo reverterá a transação caso o serviço seja transacional.

2. Código de identificação da mensagem: este código é uma seqüência de caracteres que identifica unicamente a mensagem dentro de sua classe de mensagens.

3. A classe da mensagem: esta classe indica ao mecanismo de mensagem onde buscar o texto da mensagem que se deseja exibir. Para uma melhor organização das mensagens foram definidas classes de agrupamento. Esta organização tem dois objetivos principais: descentralizar o repositório de mensagens que deve ser freqüentemente alterado pelas adições e remoções de mensagens e facilitar a integração de novos componentes na arquitetura com suas próprias classes de mensagem. Cada componente pode agrupar suas mensagens dentro de uma classe unicamente referenciada por ele. Desta forma, qualquer manutenção a ser realizada que exija a inclusão, exclusão ou alteração de uma mensagem gerada pelo componente, poderá ser completada consistentemente, pois o mesmo terá sua própria classe de mensagens que, por convenção, deve ficar no mesmo pacote do componente. Isto facilita as alterações

e, também, a integração dos componentes, evitando a entrega do componente para o ambiente de produção sem a companhia de suas mensagens, causando futuros erros quando estas mensagens forem requisitadas pela arquitetura.

4. Parâmetros da mensagem: É uma lista de objetos com informações dinâmicas que são utilizadas na composição da mensagem a ser exibida para o operador, de acordo com a execução atual do componente. Quando uma mensagem é armazenada em sua classe, podem ser definidos alguns locais na mensagem que deverão ser preenchidos com parâmetros que serão fornecidos pelo componente que gerou a mensagem. Como por exemplo: “Foram importados {0} registros com sucesso”. O componente que gerar esta mensagem deverá fornecer um parâmetro que indicará o número de registros que ele processou com sucesso.

Na arquitetura proposta, dentro de suas camadas, quando uma determinada tarefa falha, em muitos casos, todas as tarefas que originaram a chamada à tarefa que falhou serão notificadas e falharão também. Desta forma, não somente a primeira mensagem de falha deverá chegar ao operador, como também todas as demais mensagens das tarefas superiores. Assim, tem-se uma lista de mensagens que é gerada e enviada para a interface. Em vários outros casos, na arquitetura, percebeu-se que uma lista de mensagens é mais útil à arquitetura do que uma única mensagem. Foi definida então uma classe chamada *MessageList* que controla uma lista de mensagens de negócio com métodos que auxiliam a manipulação desta lista.

Integrando-se ao mecanismo de mensagens, todas as exceções foram abstraídas tendo como classe base a *BusinessException*. Basicamente, esta classe exige que em todo levantamento de exceção seja fornecida uma lista de mensagens. Esta mesma lista de mensagens pode ser utilizada pelos outros componentes em execução para adicionar novas mensagens que complementem as informações sobre a exceção ocorrida. Assim, uma lista completa das atividades que falharam será exibida na interface.

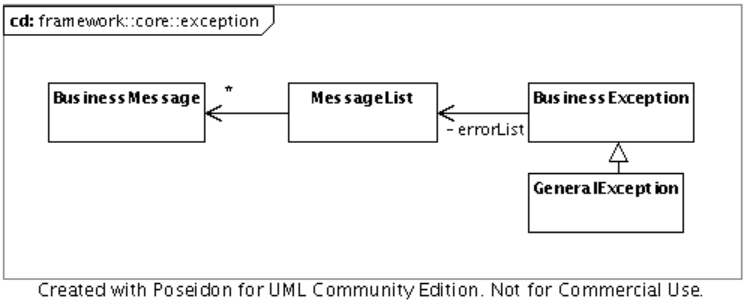


Figura 4.7: Classes que compõem o mecanismo de tratramento de mensagens e exceções.

4.2.6. O pacote de utilitários

O pacote *útil* é destinado às classes auxiliares que realizam pequenas tarefas de tratamento e formatação de dados. A maioria dessas classes foi definida para suprir necessidades de implementação do protótipo do sistema de KDD utilizando a arquitetura proposta. A Figura 4.8 mostra as classes implementadas e seus métodos.

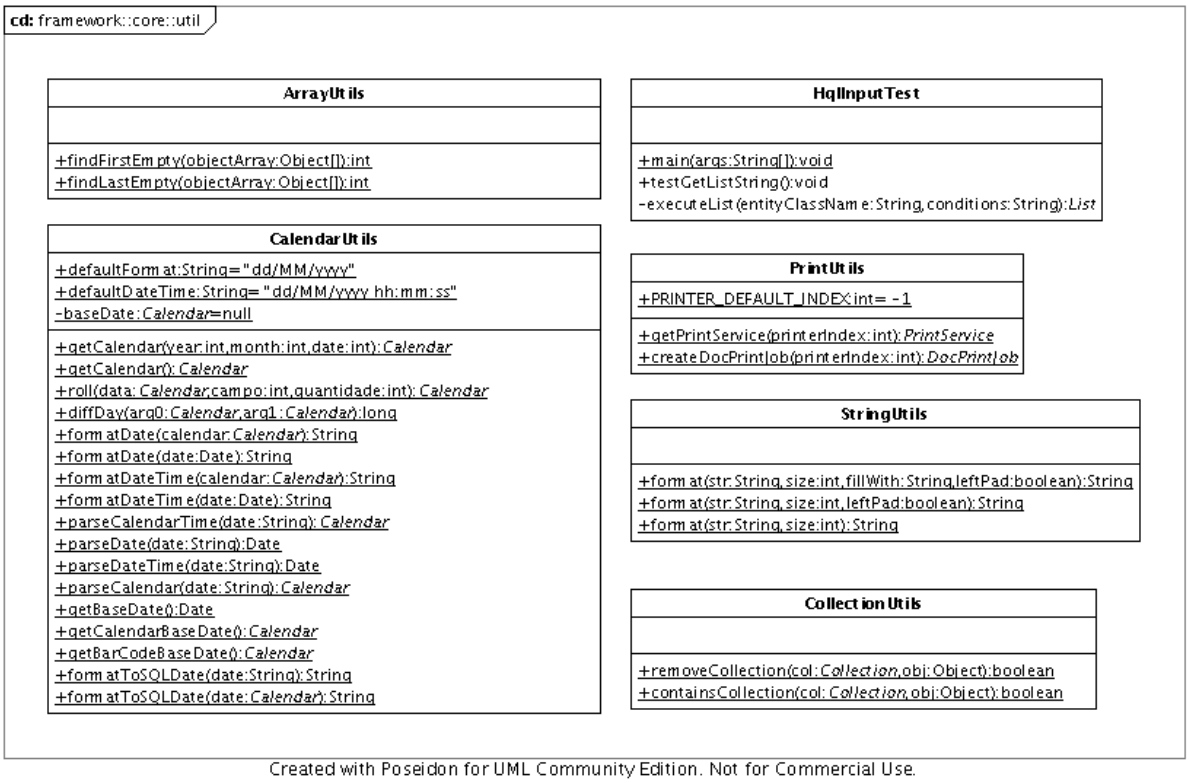


Figura 4.8: Classes do pacote Utilitário.

4.3. Demais módulos do framework

Como foi mostrado nas seções anteriores, além das camadas, a arquitetura define padrões para mensagens e para criação de testes unitários. No entanto, tudo isto ainda não é suficiente quando se trata de uma arquitetura para o desenvolvimento de sistemas de KDD. Resta ainda definir: como a arquitetura gerencia as entidades do sistema, como a arquitetura controla o acesso dos operadores e os direitos de cada operador, e por fim, como a arquitetura pode fornecer informações aos administradores do sistema para indicar o que, quando, quem e onde as atividades do sistema estão sendo disparadas. Se a proposta terminasse por aqui, ainda muitas questões ficariam para que a equipe de desenvolvimento resolvesse e mais tempo seria dedicado às questões de engenharia de software do que ao domínio KDD.

Seguindo as especificações da arquitetura de referência, esta seção apresenta as outras partes do *framework* que foram definidas para o controle de segurança, auditoria e o mecanismo CRUD.

A Figura 4.9 mostra o relacionamento entre o núcleo e os demais módulos do *framework*. Os pacotes *security*, *auditorship* e *crud* estendem o núcleo. Isto porque, internamente, estes pacotes são organizados em visualização, processos, serviços e entidades que se integram ao *framework*.

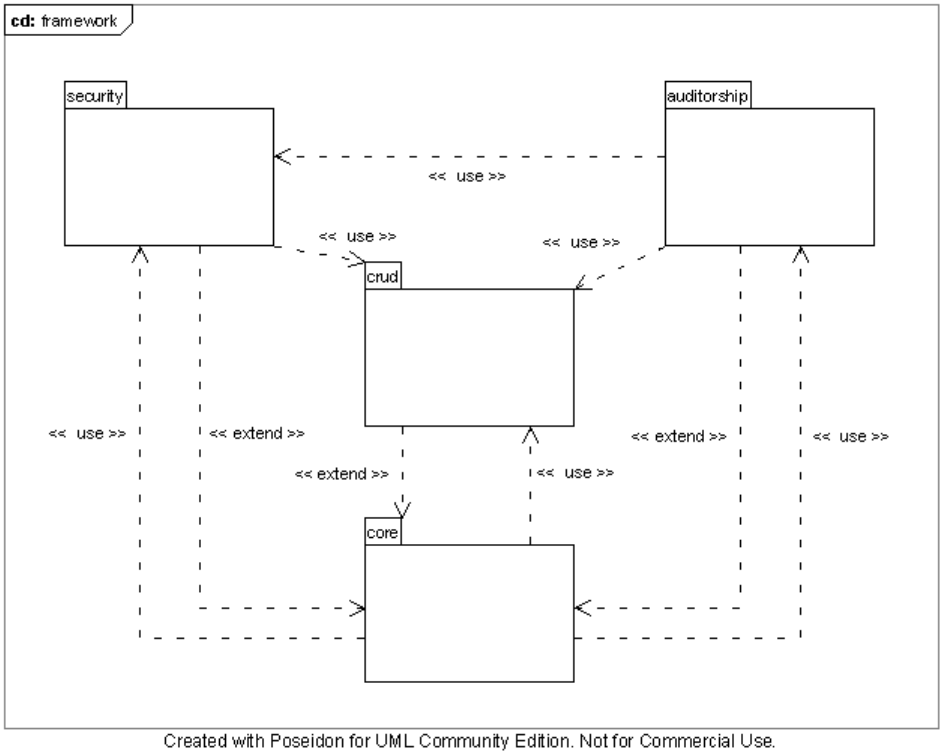


Figura 4.9: Pacotes gerais que compõem o *framework*.

4.3.1. Segurança (*Security*)

Seguindo a definição da arquitetura de referência, este pacote implementa as entidades, serviços e processos que realizam o controle de segurança do *framework*. A seguir são descritos as entidades (Figura 4.10), os serviços (Figura 4.11) e os processos (Figura 4.12) implementados.

Entidades

Os direitos de acesso do *framework* são definidos por grupo (*SecurityGroup*) e não por usuário (*ApplicationUser*). A Figura 4.10 mostra que um usuário pode se relacionar com vários grupos de segurança. Durante a verificação dos direitos do usuário são consultados todos os grupos aos quais ele pertence, se um ou mais deles atribuir o direito, então o usuário poderá acessar o recurso solicitado.

Cada grupo possui separadamente o controle dos direitos de acesso aos processos (*RightProcess*) e direitos de manipulação das entidades (*RightCrud*). Todas as entidades e processos do sistema são registrados no controle de segurança pelas classes *ApplicationEntity* e *ApplicationProcess*.

Por se tratar de um *framework* extensível, vários módulos podem ser implementados e integrados em uma única aplicação. Desta forma, o controle de segurança mantém o registro de entidades e processos separados pelos módulos (*ApplicationModule*).

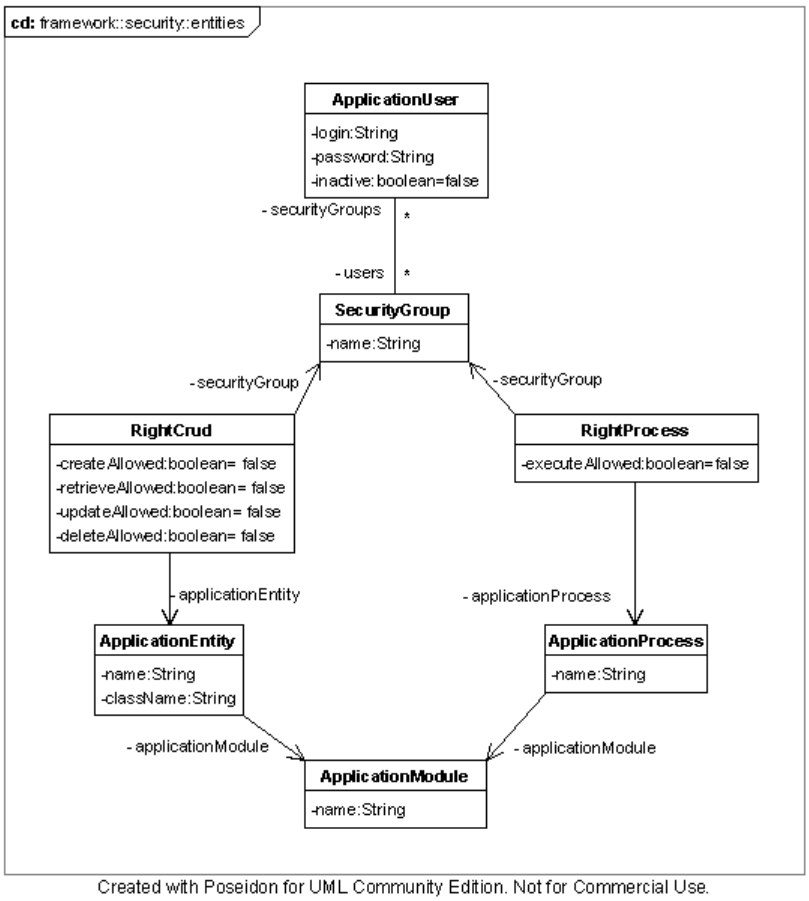


Figura 4.10: Entidades que compõem o controle de segurança.

Serviços

Para manter a estrutura de segurança, foram criados alguns serviços que facilitam a definição e a verificação dos direitos de acesso. Esses serviços são mostrados no diagrama de classe da Figura 4.11.

O serviço *ManageSecurityStructureService* é responsável por criar as estruturas de diretos para novos componentes que são integrados na aplicação. Durante sua execução, ele analisa o gerenciador de processo e o mecanismo CRUD para verificar se há um novo processo ou uma nova entidade que ainda não foi registrada na estrutura de segurança. Este procedimento automatiza a integração de novos módulos, evitando que o programador tenha que ficar alterando as estruturas de segurança toda vez que ele implementar uma nova funcionalidade.

O serviço *CreateSecurityStructureService* define a estrutura de segurança para um usuário que é passado como parâmetro. Ele é útil durante o cadastramento de novos usuários e grupos, pois permite definir de uma só vez os direitos de acesso para todos os processos e as entidades registrados.

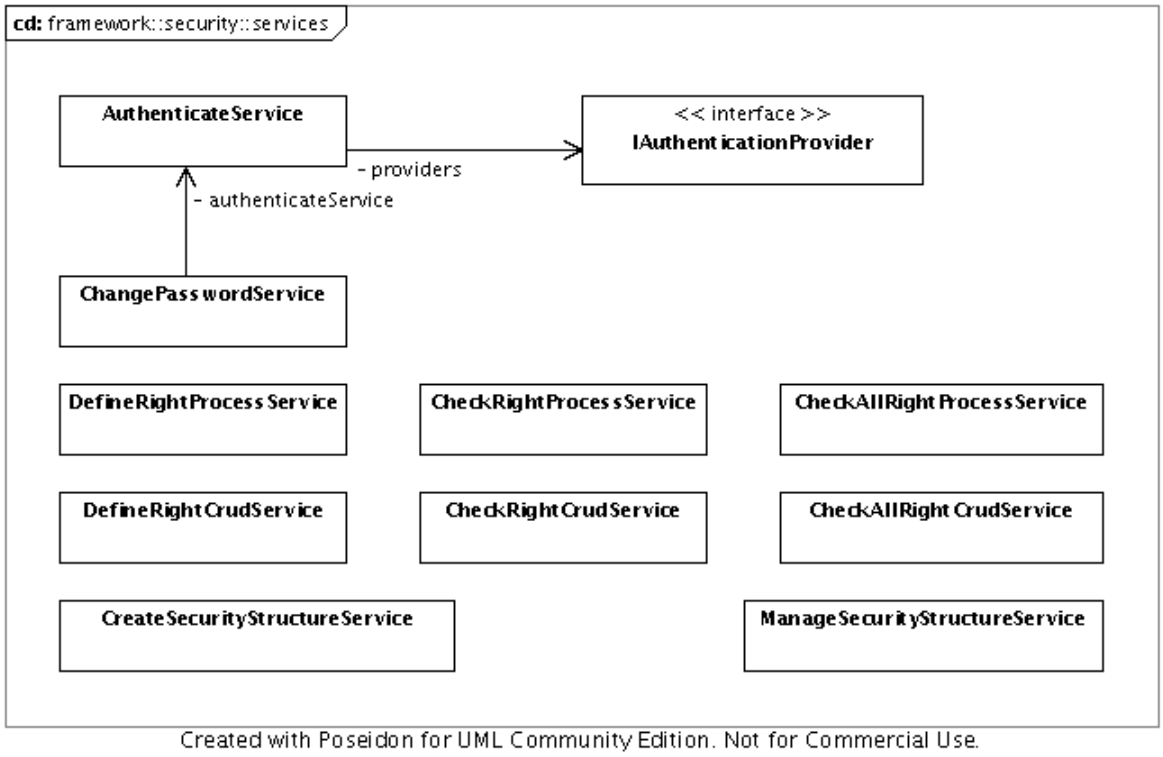


Figura 4.11: Serviços que compõem o controle de segurança.

Processos

Os processos básicos para o controle de segurança do *framework* são mostrados na Figura 4.12. O *AuthenticateProcess* permite autenticar um usuário e criar uma seção autenticada (*UserSession*) que é utilizada pelo gerenciador de processo para permitir ou negar o acesso do usuário aos processos disponíveis. O processo *ChangePasswordProcess* permite alterar a senha de um usuário. O processo *OverwritePasswordProcess* foi criado para permitir sobrescrever a senha de usuários que a tenham esquecido.

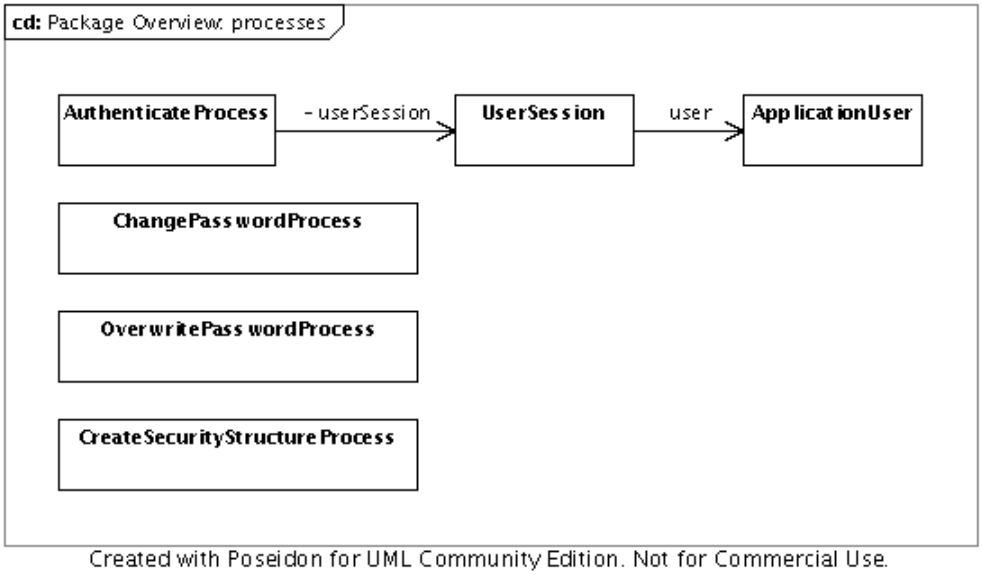


Figura 4.12: Processos do controle de segurança.

4.3.2. Auditoria

A auditoria é um controle que é utilizado para registrar informações sobre a execução de processos e de manipulação das entidades do sistema. Com este controle, é possível registrar qual dos operadores do sistema criou, alterou ou excluiu determinada entidade.

Este controle é útil para gerenciar, principalmente, as alterações das estruturas das bases de dados de origem, área de estágio, DW e do armazém de resultados. A importância deste controle é notada quando há uma equipe grande manipulando as estruturas do sistema.

O controle de auditoria, que foi integrado na arquitetura, é um complemento a mais para a segurança. Apesar dos autores de DW e KDD citarem somente o item de controle de acesso como um item de segurança, a auditoria vem adicionar informações sobre quem, onde e o que está sendo feito no do sistema.

O controle de auditoria é integrado com o controle de segurança. As entidades da aplicação, os processos da aplicação e os operadores da aplicação são utilizados nos registros da auditoria para identificar as ações e as entidades manipuladas.

Entidades

A Figura 4.13 mostra as entidades que compõem o controle de auditoria do *framework*. Como na estrutura de segurança, a auditoria de processos é separada da auditoria das entidades.

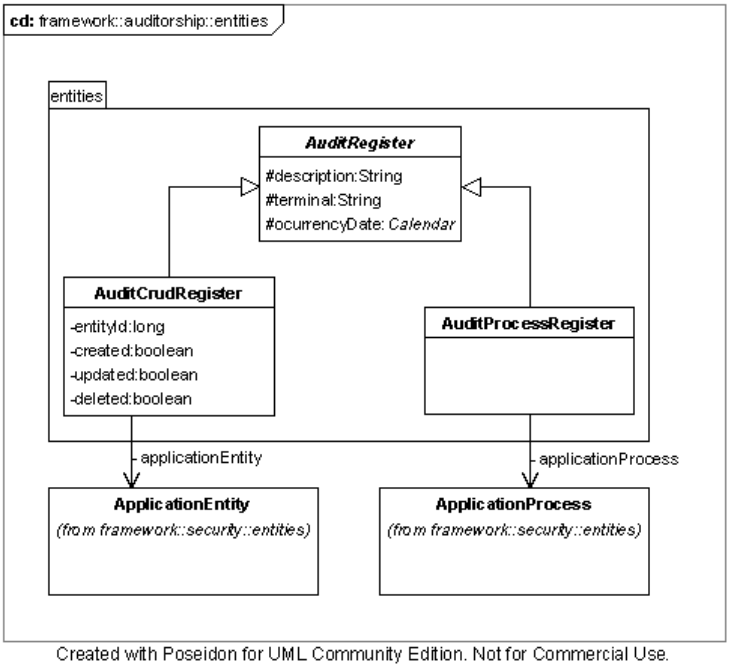


Figura 4.13: Entidades que compõem o controle de Auditoria.

Serviços

A Figura 4.14 mostra os dois serviços básicos que são utilizados para se gerar um registro na auditoria (*AuditorProcessService*) e (*AuditorCrudService*). O serviço *CheckAuditrudService* obtém todas as operações que foram realizadas em uma determinada entidade.

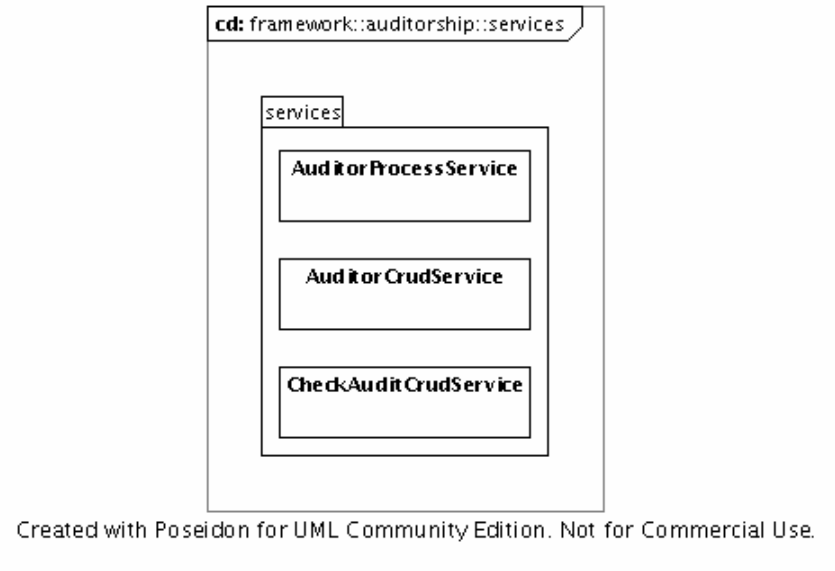


Figura 4.14: Serviços que compõem o controle de auditoria.

4.3.3. CRUD

A sigla *CRUD* é a união das primeiras letras das palavras em inglês *Create, Retrieve, Update e Delete*. Ela se refere às quatro operações básicas que são aplicadas às entidades de um sistema: criação, obtenção, alteração e remoção.

Neste trabalho, foi criado um mecanismo CRUD que é responsável pela implementação destas operações. Como já foi discutido nas seções anteriores, o CRUD possui também a responsabilidade de gerenciar a portabilidade dos dados e de juntar às entidades os metadados de validação e de descrição em uma única estrutura. Esta seção apresenta os detalhes de implementação e alguns conceitos que envolvem este mecanismo.

O conceito de entidade utilizado neste trabalho é o mesmo utilizado em desenvolvimento de sistemas de informação. Uma entidade consiste em uma classe que possui propriedades e relacionamentos com outras entidades, como mostra a Figura 4.15. Cada propriedade de uma classe possui um tipo de dado que indica o que ela armazena. Os tipos de dados mais comuns são: seqüência de caractere, número inteiro, número com ponto flutuante e datas. Cada tipo de dado pode ter algumas variantes, dependendo da linguagem de modelagem e de implementação que está sendo utilizada. No caso da linguagem Java, que é utilizada neste trabalho, tem-se os tipos de dados *String*, *Integer*, *int*, *Long*, *long*, *Float*, *float*, *Double*, *double*, *BigDecimal*, *Date* e *Calendar*.

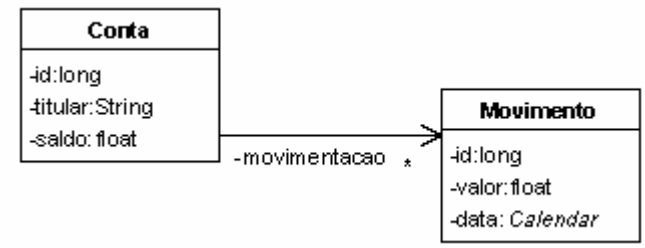


Figura 4.15: Uma entidade com propriedades de tipos primitivos e seu relacionamento com outra entidade.

Durante a implementação de uma entidade, tem-se a classe que define esta entidade, como mostra a Figura 4.15, e outras informações que geralmente são espalhadas por toda a aplicação. No exemplo desta figura são mostradas algumas propriedades. Durante a exibição dos valores da propriedade na interface com o usuário, serão acrescentadas a ela uma identificação legível e uma pequena descrição que auxilie o usuário a entender o significado da propriedade. O problema é que estas informações são necessárias em diversas interfaces

que manipulam esta entidade e, muitas vezes, elas são repetidas em cada interface que as utiliza. Caso seja necessária a alteração desta informação, todas as interfaces relacionadas deverão ser alteradas, aumentando consideravelmente o custo da manutenção. A solução é manter estas informações em um único lugar que seja de fácil acesso à aplicação. O mecanismo CRUD vai um pouco além da centralização dos metadados das entidades. Quando uma entidade é solicitada, ele a recupera do repositório e a relaciona com seus metadados, de forma que a aplicação ao mesmo tempo em que acessa uma entidade já pode acessar todos os metadados desta entidade.

Em outras palavras, o CRUD funciona como uma camada de abstração que facilita o gerenciamento das entidades de um sistema. A Figura 4.16 mostra como o mecanismo CRUD funciona.

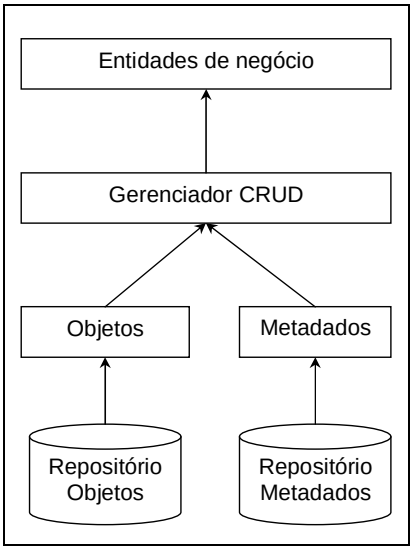


Figura 4.16: Funcionamento do mecanismo CRUD.

Uma vantagem de se ter este mecanismo é que ele facilita a implementação de máquinas que manipulam de forma automática as entidades. Além disso, é possível implementar geradores automáticos de interfaces que são capazes de identificar quantas propriedades uma entidade tem, o tipo destas propriedades, os responsáveis pela validação e, ainda, as informações textuais. As alterações nos metadados de uma entidade são automaticamente refletidas nas interfaces.

O mecanismo CRUD é um dos mais complexos implementados no *framework*. Algumas questões de internacionalização dos metadados ainda não estão eficientemente resolvidas. Na atual implementação, o gerenciador de metadados manipula somente um repositório de uma única linguagem. Outra questão é que o gerenciador armazena os metadados em arquivos, isto

dificulta um pouco a manutenção. O ideal seria utilizar um gerenciador de banco de dados para manter os metadados, isto aumentaria o número de ferramentas disponíveis que poderiam ser utilizadas para a alteração do mesmo. No entanto, para obter uma versão estável do mecanismo dentro do cronograma deste trabalho, estas questões foram deixadas para trabalhos futuros.

Entidades

A Figura 4.17 mostra uma visão geral das interfaces definidas no mecanismo CRUD. Utilizando interfaces é possível trocar as implementações sem alterar a interdependência entre os objetos. Desta forma, a equipe que utiliza o *framework* poderá escolher, por exemplo, implementar o seu próprio gerenciador de metadados. Para isto, basta implementar uma classe que satisfaça à interface *IMetadataHandle*.

Quando um objeto é obtido no repositório, o gerenciador de entidade (*IEntityManager*) obtém os metadados da entidade utilizando o gerenciador de metadados (*IMetadataHandle*) e cria uma estrutura que implementa a interface *IEntity*. Desta forma, é possível manipular os valores das propriedades do objeto usando as interfaces *IProperty* e *IPropertyValue* que são mais sofisticadas do que os métodos básicos de acesso às propriedades (*get()/set()*) implementados no objeto.

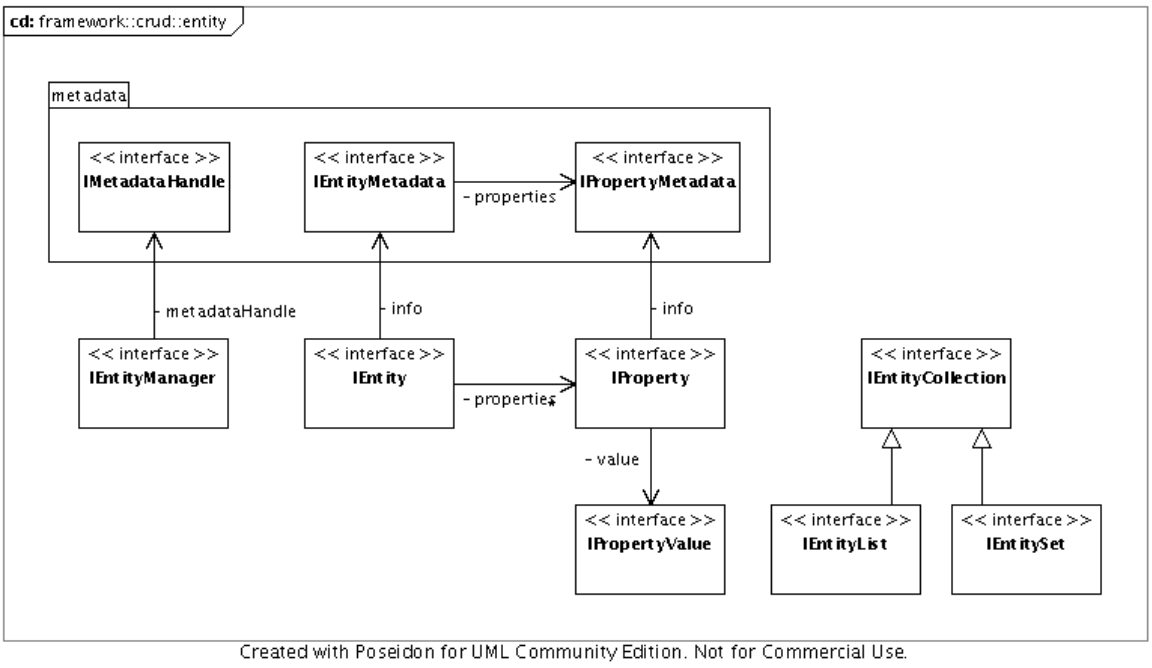


Figura 4.17: Visão geral das interfaces que compõem o mecanismo de manipulação de entidades.

A Figura 4.18 apresenta uma visão mais detalhada das interfaces que abstraem uma entidade de negócio no módulo CRUD. Observando os métodos definidos pelas interfaces é possível ter uma noção de suas funcionalidades. As interfaces *IEntityMetadata* e *IPropertyMetadata* fornecem várias informações sobre a entidade e suas propriedades.

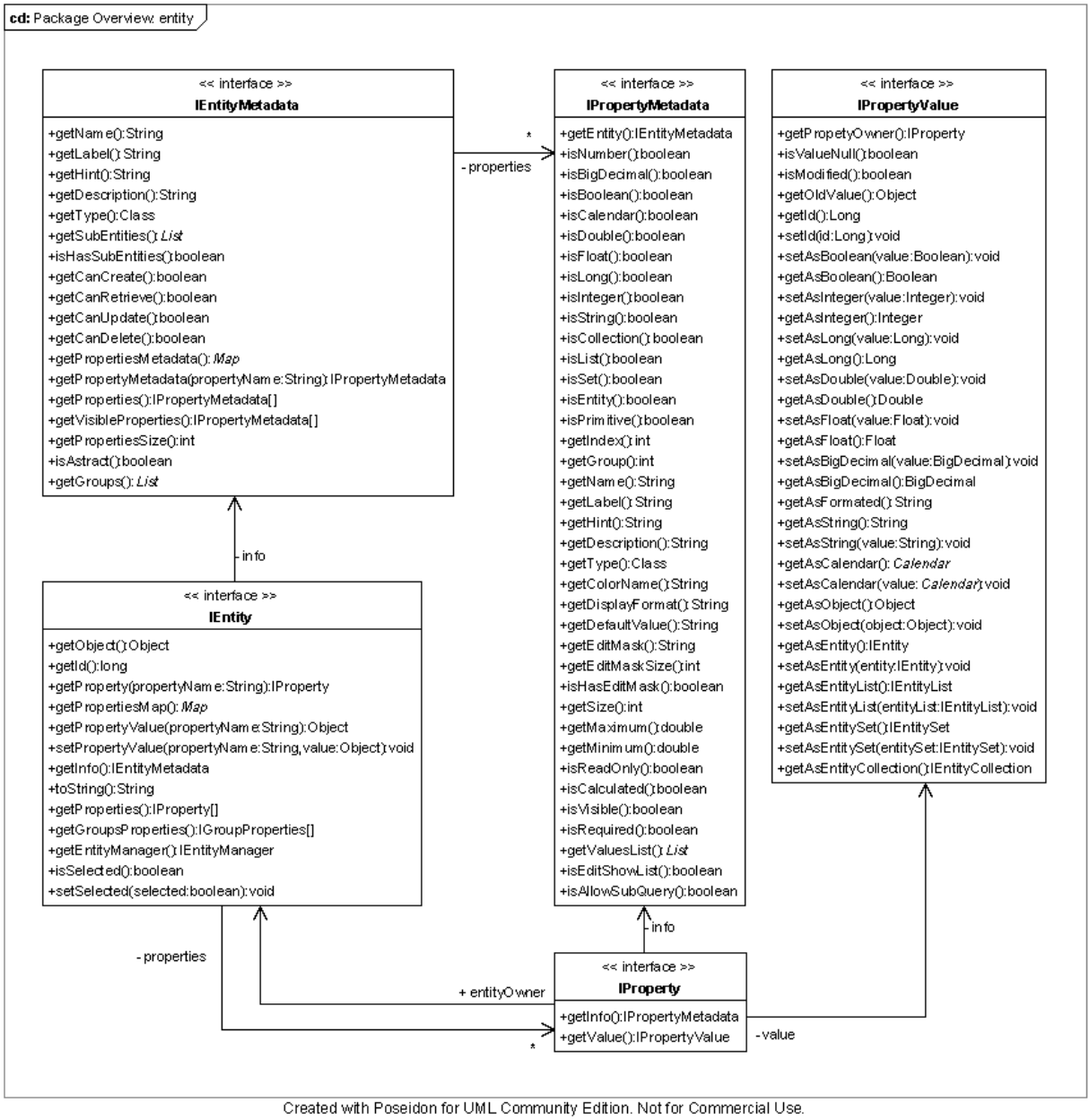


Figura 4.18: Detalhes das interfaces que abstraem as informações de uma entidade de negócio.

Serviços

A Figura 4.19 apresenta os serviços que são implementados pelo mecanismo CRUD. Todos os serviços estendem o serviço básico *CrudServiceBasic* que implementa as funcionalidades

básicas para o acesso aos objetos persistidos. Os serviços *GetAllEntitiesService* e *GetCrudEntitiesService* são responsáveis pela criação de uma lista das entidades que estão atualmente registradas na execução da aplicação. Estes serviços são utilizados pelo mecanismo de segurança para determinar quais entidades devem ser cadastradas em suas estruturas.

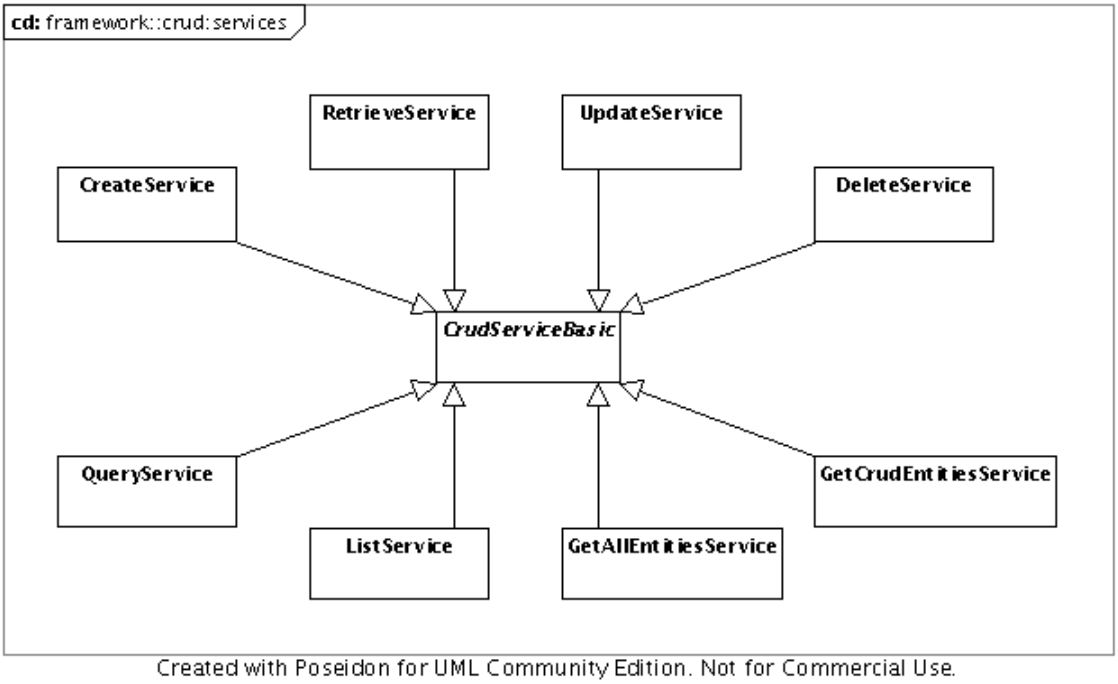


Figura 4.19: Serviços do mecanismo *CRUD*.

Processos

A Figura 4.20 mostra os processos que implementam as operações *CRUD*. O processo *QueryProcess* oferece recursos para a realização de pesquisas avançadas no repositório de objetos.

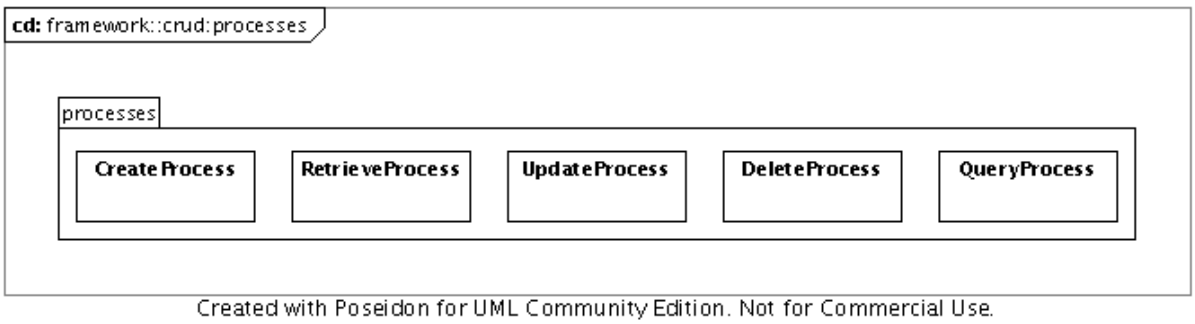


Figura 4.20: Processos que implementam as operações *CRUD*.

4.4. Considerações Finais

A implementação do *framework* foi um desafio para este trabalho, isto porque as questões que envolvem o desenvolvimento de um software corporativo vão mais além do que somente a especificação de sua arquitetura. Como foi descrito, muitas questões tiveram que ser analisadas e a busca pelas respostas, apesar de ser apresentada em poucas páginas, demandou muito tempo e paciência.

As questões que foram respondidas para a construção do *framework* são perfeitamente aplicáveis ao desenvolvimento de aplicações para muitos outros domínios. O próprio *framework* foi implementado de uma maneira que não fosse dependente do domínio de sistemas KDD, apesar de cumprir com os requisitos levantados nos capítulos anteriores.

A organização do *framework* mostra-se como uma estrutura bem coesa e de fácil manutenção e extensão. O mecanismo *CRUD*, definido e implementado por este trabalho, possui um caráter inovador de junção de objetos com metadados para a obtenção de entidades de negócio mais completas e complexas.

Um aspecto interessante de se ressaltar é que mesmo com a arquitetura bem definida, a atual versão de implementação do *framework* é bastante diferente das anteriores. Com o decorrer do tempo, a visão sobre a própria arquitetura e sua implementação foi ficando mais clara. No início da definição deste trabalho havia um abismo entre a arquitetura inicial proposta e sua real implementação. Com o decorrer das pesquisas, foram ocorrendo sinapses de idéias. A arquitetura inicial foi completamente alterada para chegar ao modelo de referência.

Talvez estas experiências possam ser mais bem detalhadas em outros documentos escritos sobre este trabalho.

5. – Estudo de Caso

Nos capítulos anteriores foram mostradas a arquitetura de software do sistema KDD e a implementação do framework que dá suporte à arquitetura de referência. Este capítulo mostra a realização de um estudo de caso utilizando um protótipo de sistema KDD implementando a arquitetura de software mostrada neste trabalho. Neste protótipo foram implementadas as funcionalidades básicas de um sistema KDD tendo como base a arquitetura de referência e a arquitetura de software propostas neste trabalho.

Não é objetivo deste trabalho apresentar uma completa implementação de um sistema KDD e com avançadas funcionalidades, mas oferecer uma arquitetura que permita a implementação e integração dessas.

Para a realização deste estudo de caso foram utilizadas duas bases de dados reais de uma empresa. As bases pertencem a dois sistemas operacionais que estão em operação normal.

Para deixar mais clara a apresentação do estudo de caso realizado, as telas de execução que mostram os recursos implementados no protótipo do sistema KDD e sua descrição funcional foram inseridas no Anexo A deste trabalho.

Este capítulo foi dividido em seções que mostram a análise do estudo de caso, a definição das origens dos dados, a definição da área de estágio, a definição do DW, a realização de operações de OLAP e o armazenamento dos resultados de OLAP.

5.1. Análise

As bases utilizadas neste estudo de caso pertencem a um sindicato que se encontra em fase de transição do seu sistema gerencial. Com isto, existem dois sistemas operacionais produzindo dados. Um sistema, aqui denominado A, é responsável pelo controle de mensalidades sociais cobradas dos associados do sindicato e o outro sistema, aqui denominado B, pelo controle do recolhimento da contribuição sindical. Ao final do período de transição, o sistema B será responsável por todos os controles internos do sindicato.

O objetivo deste estudo de caso é integrar as bases de dados dos sistemas A e B, extraindo as movimentações financeiras dos dois sistemas e as armazenando na Área de Estágio, permitindo que sejam executadas operações de OLAP ou de MD nos dados integrados.

Através do uso das bases de dados dos sistemas A e B, são mostradas a integração das bases dentro do sistema KDD e as funcionalidades implementadas no protótipo.

Para o estudo de caso foram usadas as tabelas *movimentacao* do sistema A e *financeiro_movimento* do sistema B. A Figura 5.1 mostra a equivalência entre os itens de dado que serão extraídos das duas tabelas. Durante a extração e a transformação, os dados equivalentes são homogeneizados e armazenados na área de estágio utilizando uma mesma representação.

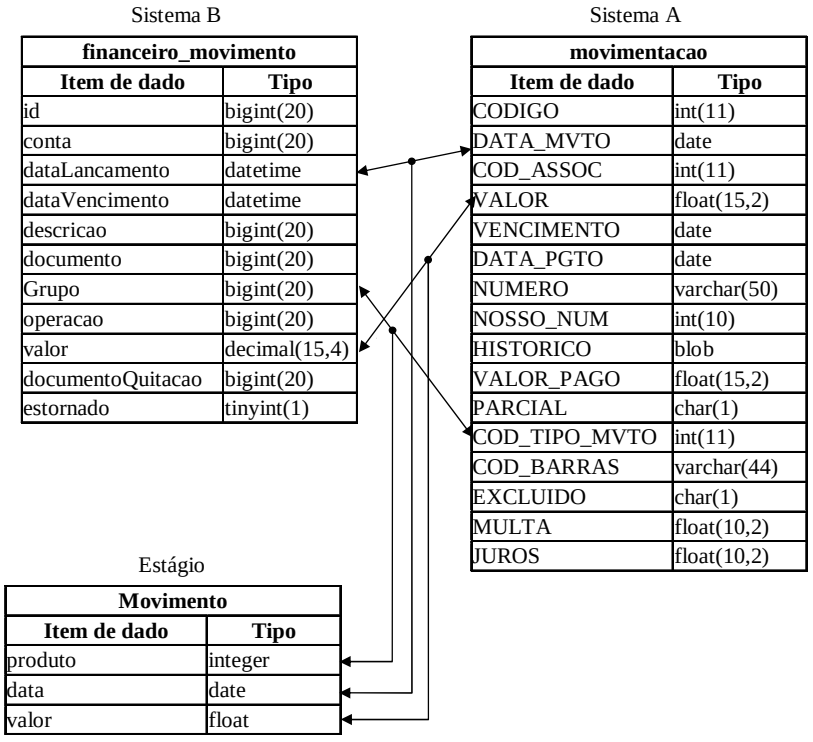


Figura 5.1: Relacionamento entre as duas bases de movimento dos sistemas A e B e a Área de Estágio.

A Figura 5.2 mostra o plano de execução das atividades de integração dos dados no sistema KDD e os repositórios de dados usados. As setas representam o fluxo dos dados e os rótulos indicam os processos que são utilizados para mover os dados de um repositório para o outro.

De acordo com a Figura 5.2, é necessário definir as conexões com os sistemas A e B, as operações de extração e transformação dos dados de origem, a estrutura da área de estágio, as operações de povoamento, a estrutura do DW e a conexão do armazém de resultados.

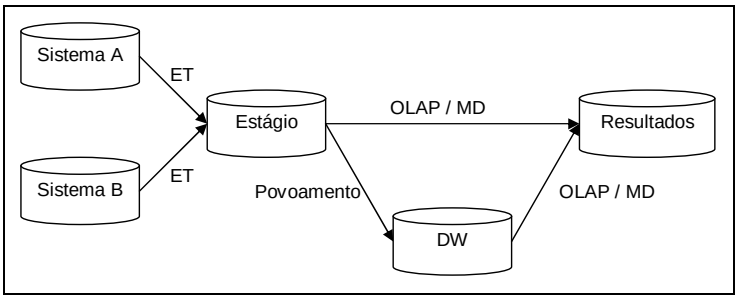


Figura 5.2: Plano de execução da integração dos dados nas bases do sistema KDD.

Neste estudo de caso, as operações de povoamento foram somente cadastradas no sistema KDD, porém não foram executadas porque os serviços de povoamento não estão implementados no protótipo. Da mesma forma, as operações de MD também não foram executadas porque é necessária a implementação das técnicas de mineração para serem utilizadas dentro do protótipo.

O protótipo implementa um processo simplificado para entrada de operações de OLAP, visualização dos resultados e armazenamento desses resultados.

5.2. Origem

Para acessar os dados das bases de origem foram definidas duas conexões de dados cujas informações são mostradas na Tabela 5.1. Neste estudo de caso, ambas as bases estão fisicamente armazenadas em um mesmo gerenciador de banco de dados. No entanto, poderiam estar em qualquer outro gerenciador, inclusive de tecnologias diferentes. Isto é possível pela utilização da ferramenta de integração de dados JDBC que é utilizada no protótipo.

Tabela 5.1: Dados das conexões com os bancos de dados de origem.

Sistema	Servidor	Esquema	Porta	URL JDBC
A	sivamar.homeip.net	sindicato	3306	jdbc:mysql://sivamar.homeip.net:3306/sindicato
B	sivamar.homeip.net	orionsigs	3306	jdbc:mysql://sivamar.homeip.net:3306/orionsigs

Utilizando as conexões de origem e o processo de definição dos itens de dados de origem, foram definidos como disponíveis para o sistema KDD os itens de dados *DATA_MVTO*, *COD_TIPO_MOVT* e *VALOR* da tabela *movimentação* do Sistema A e os itens de dados *dataLancamento*, *grupo* e *valor* da tabela *financeiro_movimento* do Sistema B.

5.3. ET e Área de Estágio

Na área de estágio foi definida uma tabela chamada *Movimento* com três itens de dados, como mostrado na Figura 5.1. Para esta definição foram utilizadas as entidades apresentadas na arquitetura de software do sistema KDD, mostradas na Figura 3.14.

Para cada item de dado foi definida uma configuração de extração e transformação. A Tabela 5.2 mostra esta configuração. Neste estudo de caso nenhuma função de transformação foi definida para ser aplicada durante a extração, somente uma carga simples é realizada. Isto porque, a implementação de algumas funções de transformação demandaria tempo. Foi mantido o foco em definir as estruturas que suportassem a implementação e a especificação destas funções.

Tabela 5.2: Configuração de extração e transformação.

Data da próxima carga desta definição	Freqüência de execução	Item de dado de origem
		Item de dado de destino
12/10/2006	Diária	sivamar.homeip.net:orionsigs.movimentacao.DATA_MVTO
		localhost:staging.Movimento.DATA
12/10/2006	Diária	sivamar.homeip.net:orionsigs.movimentacao.VALOR
		localhost:staging.Movimento.VALOR
12/10/2006	Diária	sivamar.homeip.net:orionsigs.movimentacao.COD_TIPO_MVTO
		localhost:staging.Movimento.PRODUTO
12/10/2006	Diária	sivamar.homeip.net:sindicato.financeiro_movimento.dataLancamento
		localhost:staging.Movimento.DATA
12/10/2006	Diária	sivamar.homeip.net:sindicato.financeiro_movimento.grupo
		localhost:staging.Movimento.PRODUTO
12/10/2006	Diária	sivamar.homeip.net:sindicato.financeiro_movimento.valor
		localhost:staging.Movimento.VALOR

Durante a realização deste trabalho, a movimentação do sistema A possuía 111.534 registros e a do sistema B possuía 13.077 registros. Para a realização deste estudo de caso foi analisada somente a movimentação do ano de 2006, o que resultou na seleção de 12.720 registros do sistema A e os 13.077 registros do sistema B.

5.4. DW

Apesar das operações de povoamento não estarem funcionais no protótipo, as estruturas do DW foram projetadas e cadastradas no sistema KDD. A Figura 5.3 mostra a matriz de barramento com o fato e as dimensões definidas e a Tabela 5.3 mostra as propriedades de cada tabela.

Dimensões / Fato	Tempo	Produto
Movimento	X	X

Figura 5.3: Matriz de barramento.

Tabela 5.3: Propriedades das tabelas de fato e de dimensões.

Nome da tabela	Propriedades
DimensaoProduto	nome (VARCHAR(100)), codigo (VARCHAR(10))]
DimensaoTempo	data (DATE), mes (INTEGER), dia (INTEGER), semestre (INTEGER), ano (INTEGER), estacao (INTEGER)
FatoMovimento	valor (FLOAT)

A questão da construção de um DW com *Data Marts* integrados é um critério que o analista pode definir. A atual estrutura de metadados não possui propriedades específicas que possam facilitar no gerenciamento dos *Data Marts*. No entanto, esta nova funcionalidade pode ser implementada pela equipe que utiliza os conceitos e as estruturas da arquitetura de referência e da arquitetura de software.

5.5. Olap

A Figura 5.4 mostra a tela de execução de uma operação de OLAP sendo realizada em uma conexão da área de estágio. Neste exemplo, está sendo pesquisado o dia do mês que possui o maior número de movimentações registradas no primeiro semestre. Vale lembrar que este resultado foi obtido dos dados dos sistemas A e B que foram integrados na área de estágio.

O resultado obtido pode ser salvo no armazém de resultados, para isto basta clicar no *link* ‘Salvar resultados na área de resultados’. A Figura 5.5 mostra a tela exibida para a escolha da conexão do armazém de resultados, o preenchimento do nome do resultado e uma descrição textual sobre o resultado.

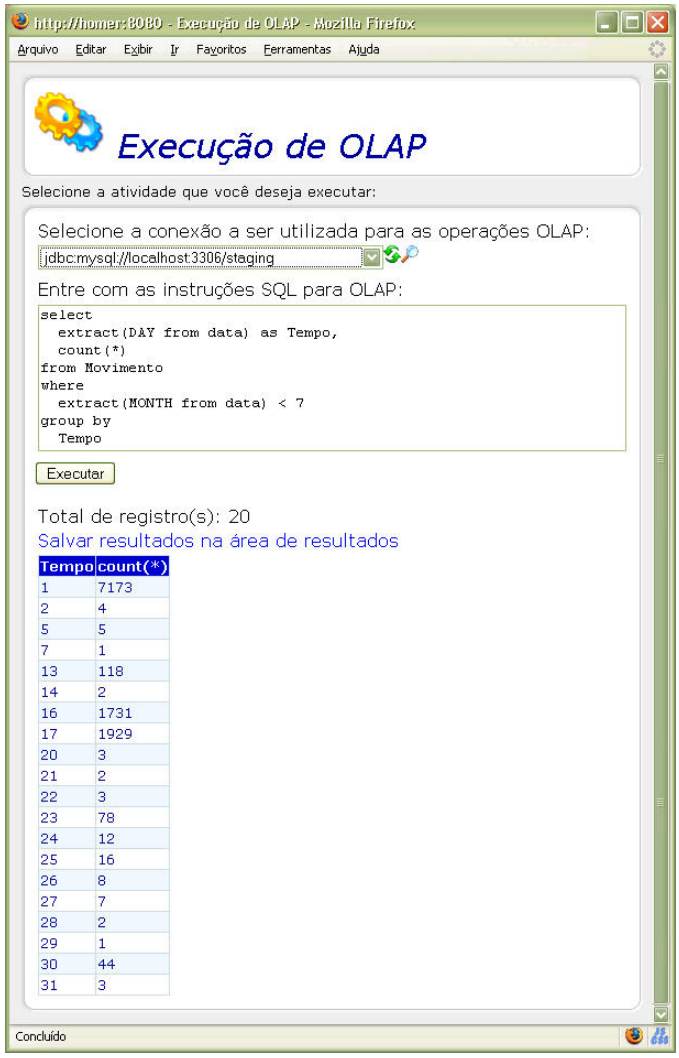


Figura 5.4: Análise da movimentação do primeiro semestre de 2006.

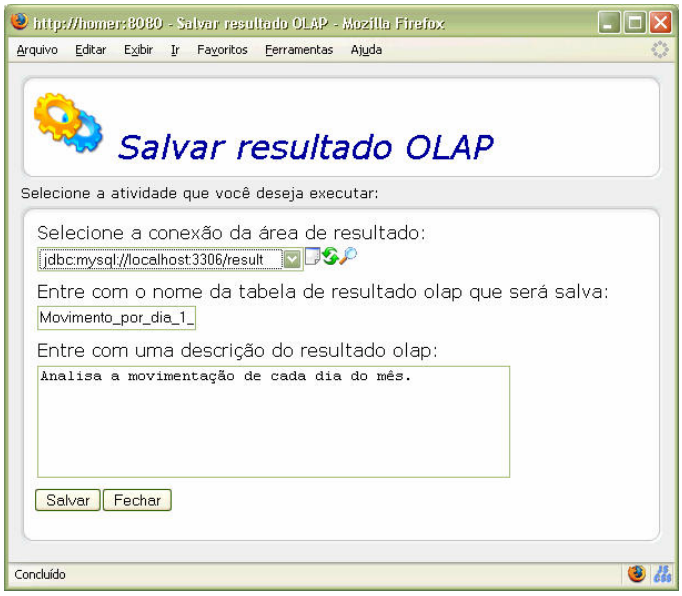


Figura 5.5: Tela de parâmetros para salvar o resultado de operações de OLAP .

Com os dados integrados, outras consultas sobre a movimentação da empresa foram realizadas utilizando a atual implementação do módulo OLAP do protótipo. Os resultados obtidos foram armazenados no armazém de resultados e podem ser consultados utilizando o próprio módulo OLAP, uma vez que o modelo de referência prevê esta operação.

5.6. Considerações finais

Com a execução deste estudo de caso, foi possível verificar a funcionalidade dos componentes de software que foram implementados tendo como base a arquitetura de referência, a arquitetura de software e o *framework*, apresentados neste trabalho.

Outras possibilidades podem ser exploradas neste estudo de caso como, por exemplo, a integração e a análise das pessoas cadastradas nos dois sistemas. No entanto, para manter o cronograma e o foco do trabalho foram considerados somente os aspectos apresentados neste estudo de caso.

Durante a implementação do protótipo, o *framework* mostrou-se bastante robusto, tratando as exceções ocorridas e exibindo mensagens amigáveis na interface. Além disso, os gerenciadores de processos e de serviços executaram todas as suas atividades sem deixar o sistema instável nenhuma vez.

Há vários pontos que precisam ser melhorados, ou ainda, implementados. O módulo de integração de ferramentas de visualização de resultados é um deles. A interface WEB do protótipo não se mostrou muito ágil, até mesmo pela sua natureza de requisição e resposta. A utilização da tecnologia Ajax[1] poderá trazer algumas vantagens para a interface.

Contudo, os resultados obtidos estão de acordo com as expectativas.

[1] http://www.ajaxmatters.com/

Conclusão

A tecnologia da informação já é uma realidade em praticamente todos os setores da sociedade. Boa parte das rotinas governamentais já é tratada por sistemas de informação. A próxima fase é a transformação de todas essas informações em conhecimento.

No entanto, ainda existem dificuldades no desenvolvimento de sistemas para busca de conhecimento devido, principalmente, à complexidade desses sistemas e à grande variedade de tecnologias envolvidas. Assim, este trabalho teve como principal preocupação a definição de uma arquitetura de software que possa ser usada na construção desses tipos de sistema.

Seguindo o padrão de construção de arquitetura apresentada por Bass et al (2003), neste trabalho foram definidos um modelo de referência, de uma arquitetura de referência e uma arquitetura de software para sistemas KDD.

Durante o desenvolvimento deste trabalho várias questões foram levantadas e as soluções alcançadas mostraram uma qualidade satisfatória. Todo o processo da concepção do modelo de referência e sua influência na definição da arquitetura de referência foram discutidos detalhadamente. Cada requisito não-funcional do modelo de referência foi mapeado para uma funcionalidade atendida pela arquitetura de referência. Ao final, a arquitetura de referência definida propõe soluções que podem ser perfeitamente utilizadas em outros domínios.

A metodologia apresentada por Merson (2006) permitiu mostrar, de forma clara, os aspectos estáticos e dinâmicos da arquitetura de referência utilizando os conceitos de visões. E, para complementar, os detalhes de implementação mostrados no capítulo do *framework* esclarecem algumas dúvidas sobre o funcionamento interno de cada componente.

Vale ressaltar que a implementação do *framework* foi um desafio para este trabalho, isto porque o desenvolvimento de um software corporativo envolve outras questões além da especificação de sua arquitetura.

Devido ao escopo e ao cronograma deste trabalho, foram propostos componentes para arquitetura de software que atendam às funcionalidades mais básicas de um processo KDD. No entanto, foi mostrado que, utilizando a arquitetura de referência, podem ser construídos e integrados componentes avançados para suportar todas as fases do processo KDD. Desta

forma, é possível visualizar como os quadros e as linhas do modelo de referência são transformados em pacotes, classes e linhas de código.

O mecanismo *CRUD*, definido e implementado neste trabalho, possui um caráter inovador de junção de objetos com metadados para a obtenção de entidades de negócio mais completas e complexas. Outro aspecto é que todos os dados sobre o processo KDD podem ser abstraídos em entidades e serem persistidos para sua reutilização e automatização das atividades de sistemas KDD.

Um aspecto interessante de se ressaltar é que com o decorrer do tempo, a visão sobre a própria arquitetura e sua implementação foi ficando mais clara. No início da definição deste trabalho, havia um abismo entre a arquitetura inicial proposta e sua real implementação. Com o decorrer das pesquisas, foram ocorrendo sinapses de idéias. A arquitetura inicial foi sendo alterada para atender aos requisitos que iam sendo identificados no modelo de referência.

Por fim, este trabalho cumpriu com sua proposta inicial de unir conceitos de descoberta de conhecimento com os conceitos de engenharia de software para a composição de um sistema *arquiteturalmente* bem fundamentado.

Trabalhos futuros

O foco principal deste trabalho está na definição de uma arquitetura de software proposta e implementada. No entanto, durante a especificação da arquitetura alguns pontos mostraram potenciais para a realização de pesquisas mais aprofundadas. Estes pontos são relacionados nesta seção, como trabalhos futuros.

Somente os princípios básicos de SOA foram aplicados neste trabalho. Questões de contratos formais e distribuição dos serviços não foram abordadas. A aplicação desses conceitos tornaria a execução de serviços mais sofisticada.

Durante a implementação do *framework* percebeu-se que, além das entidades de negócio, os processos de negócio também devem conter informações textuais sobre sua funcionalidade e seus parâmetros. Estas informações estariam no repositório de metadados e serviriam para os controladores de visualização exibi-las para o usuário.

A portabilidade de interface tratada pela arquitetura de referência deverá ser mais bem explorada com o desenvolvimento de sistemas que utilizam outra tecnologia de interface que não seja a WEB.

Pela atual proposta, todos os armazéns de dados do sistema KDD (Fontes Operacionais, Área de Estágio, Data Warehouse e Armazém de Resultados) são prováveis fontes de dados para as operações de OLAP e de MD, como é apresentado no modelo de referência da Figura 3.1. Com isto, falta discutir melhor sobre a questão do armazenamento de dados históricos na área de estágio e no armazém de resultados. Nas atuais literaturas a área de estágio é considerada uma área temporária dos dados e o DW uma área histórica. De qualquer forma, a equipe que usará a arquitetura proposta poderá especificar se utilizará ou não a área de estágio para operações OLAP e de MD.

Referências bibliográficas

Appice, A., Ceci, M., Malerba, D., ***KDB2000: An integrated knowledge discovery tool***. Dipartimento di Informatica,University of Bari, Italy, http://www.di.uniba.it/~malerba/#Software, 2006.

Ballard, C ; Herreman, D.; Schau, D.; Bell, R.; Kim, E.;Valencic, A. ***Data Modeling Techniques for Data Warehousing.*** IBM Corporation, 1998.

Bass, L., Clements, P., Kazman, R., ***Software Architecture in Practice***. Second Edition. Boston, MA: Addison-Wesley, 2003.

Bass, L., Klein, M., Bachmann, F., ***Quality Attribute Design Primitives***. CMU/SEI-2000-TN-017, 2000.

Bauer, C., King, G., ***Hibernate in action***. Manning Publications Co., 2005.

Braga, R. T. V., ***Um Processo para Construção e Instanciação de Frameworks baseados em uma Linguagem de Padrões para um Domínio Específico***. Instituto de Ciências Matemáticas e de Computação- ICMC/USP, Tese de Doutorado, Universidade de São Paulo – SP, 2002.

Brezany, P., Hofer, J., Tjoa, A., Wöhrer, A., ***Gridminer: An Infrastructure for Data Mining on Computational Grids***. APAC'03 Conference, Gold Coast, Australia, October 2003

Broemmer, D., ***J2EE Best Practices - Java Design Patterns, Automation, and Performance***. Wiley Publish Inc, EUA, 2003.

Buschmann, F.; Meunier, R.; Rohnert, H. et al., ***Pattern-Oriented Software Architecture - A System of Patterns***. Wiley & Sons Ltd., 1996

Castoldi, A. V., ***Um Modelo de Meta Dados para Suporte a Sistemas de Descoberta de Conhecimento em Bancos de Dados***. Trabalho de Graduação. Universidade Estadual de Maringá. Maringá, Paraná, 1999.

Chaudhuri, S.; Dayal, U., ***An Overview of Data Warehousing and OLAP Technology***. 2004.

Cherobino, V., ***A evolução do ambiente de TI***. Partners Ecosystem, ano 1, Número. 2 de Julho, Agosto e Setembro de 2006.

Coad, P., ***Object-Oriented Patterns***. Communications of the ACM, V.35, nº9, p. 152-159, setembro 1992

Coplien, J.; Schmidt, D. ***Pattern Languages of Program Design***. Reading-MA, Addison-Wesley, 1995.

David, M. ***Building and managing the meta data Repository: A Full Lifecycle Guide***, Paperback, 2001.

Dias, M.M., ***Um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados***. Tese de Doutorado. Universidade Federal de Santa Catarina. Florianópolis, Santa Catarina, 2001.

Fayyad, U. M., Piatesky-Shapiro,G.; Smyth, P. e Uthurusamy, R., ***From Data Mining to Knowledge Discovery in Database***. Cambridge, AAAI/MIT Press,1996.

Gallagher, B. P., ***Using the Architecture Tradeoff Analysis Method to Evaluate a Reference Architecture: A Case Study***. Nota técnica, CMU/SEI-2000-TN-007, 2000.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. ***Design Patterns -Elements of Reusable Object-Oriented Software***. Reading-MA, Addison-Wesley, 1995.

Gupta, S. K.; Bhatnagar, V.; Wasan, S.K., ***Architecture for knowledge discovery and knowledge management***. Knowledge and Information Systems, 2004.

Han, J., Kamber, M., ***Data Mining: Concepts and Techniques***. The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor Morgan Kaufmann Publishers, 2001.

Herden, A., ***UPKDD – Um Processo para Desenvolvimento de Sistemas de Descoberta de Conhecimento em Banco de Dados***. Proposta de dissertação de mestrado, Departamento de Informática da Universidade Estadual de Maringá, 2006.

Inmon, W. H. ***Como Construir o Data Warehouse.*** Rio de Janeiro, 1997.

Jacobson, I., Booch, G., ***The Unified Software Development Process***. Editora Addison Wesley, 1998.

Johnson, R. E., Foote, B., ***Designing Reusable Classes***. Journal of Object Oriented Programming – JOOP, 1(2):22-35, Junho/Julho 1988.

Kazman, R., Klein, M., Clements, P., ***ATAM: Method for Architecture Evaluation,*** Technical Report, CMU/SEI-2000-TR-004, 2000.

Kimball R.; Reeves L.; Ross M.; Thornthwaite W., ***The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses***, John Wiley & Sons Inc., 1998.

Martin, R.C.; Riehle, D.; Buschmann, F. ***Pattern Languages of Program Design 3***. Reading-MA, Addison-Wesley, 1998.

Menolli, A. L. A., ***Definição de uma Arquitetura de Data Warehousing para Gestão em Ciência e Tecnologia no Brasil***. Tese de Mestrado. Universidade Estadual de Maringá. Maringá, Paraná, 2004.

Merson, P., ***Mini-Curso: Como Documentar Arquitetura de Software,*** disponível no endereço http://www.sei.cmu.edu/ata, 2006.

Metsker S. J., ***Design Pattern Java Workbook***. Addison Wesley, 2002.

Methanias, J., ***O Data Warehouse não foi e nem é o problema***. Disponível no endereço http://www.sqlmagazine.com.br/Colunistas/Methanias/03_DataWareHouse.asp. Consultado em outubro de 2004.

Murphy, C., METHODS & TOOLS - Global knowledge source for software development professionals. ISSN 1023-4918, Volume 13 - number 1, Spring 2005;

O'Brien L., Bass, L., Merson, P. ***Quality Attributes and Service-Oriented Architectures***. CMU/SEI-2005-TN-014

OASIS Standard, ***Reference Model for Service Oriented Architecture 1.0.*** OASIS (Organization for the Advancement of Structured Information Standards). Disponível no

endereço http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html. Consultado em setembro de 2006.

Oracle, ***Oracle Warehouse Builder Documentation***. Disponível no endereço http://www.oracle.com/technology/documentation/warehouse.html. Consultado em junho de 2005.

Pamplona, V. F., ***Tutorial Java: O que é Java?,*** disponível no endereço http://www.javafree.org/content/view.jf?idContent=84, acessado em setembro de 2006.

Pinheiro, C. A. R., ***Gestão do conhecimento***. http://www.inf.ufrgs.br/~wives/portugues/projeto.html. Consultado em junho de 2005.

Sell, D., ***Uma Arquitetura para Distribuição de Componentes Tecnológicos de Sistemas de Informações Baseadas em Data Warehouse***. Programa de Pós-Graduação em Engenharia de Produção, Florianópolis – SC. Dissertação de Mestrado, UFSC, Florianópolis, 2001.

Shaw, M.; Garlan, D., ***Software Architecture – Perspectives on an Emerging Discipline***. Prentice Hall, Upper Saddle River, New Jersey, 1996.

Singh, H. S., ***Data Warehouse: Conceitos, Tecnologias, Implementação e Gerenciamento***. Makron Books, 2001.

Tait, T. F. C, ***Um Modelo de Arquitetura de Sistemas de Informação para o Setor Público: estudo em empresas estatais prestadoras de serviços de informática***. Tese de Doutorado. Universidade Federal de Santa Catarina. Florianópolis, Santa Catarina, 2000.

Tate, B. A., ***Bitter Java***, Manning Publications Co. - Greenwich, EUA, 2002.

Ullman, J. D., ***Principles of Database and Knowledge-Based Systems – Volume I: Classical Database Systems***. Standford University. Computer Science Press, 1988.

Vassiliadis, P., ***Gulliver in the land of data warehousing: practical experiences and observations of a researcher***. Database Systems Laboratory, Zografou 15773, Athens, Greece. 2000.

Vlissides, J.; Coplien, J.; Kerth, N, ***Pattern Languages of Program Design 2***. Reading-MA; Addison-Wesley, 1996.

A. ANEXO - Telas de execução do estudo de caso

Os passos a seguir mostram a execução das atividades para a integração dos dados no sistema KDD e as ferramentas implementadas no protótipo.

A Figura A.1 mostra a tela de autenticação do operador. O operador deve preencher os campos solicitados e clicar no botão *Entrar*. Internamente, o controlador desta visão irá acionar o processo de autenticação e repassar-lhe os dados, como é mostrado na Figura 3.7.

Se a autenticação for executada com sucesso o usuário visualizará a tela inicial do sistema KDD, mostrada na Figura A.2. Esta tela está dividida em três partes: cabeçalho, corpo e rodapé.

No cabeçalho são exibidas as informações sobre a estação que está acessando o sistema e o operador autenticado. Nesta parte também é possível acessar o menu de Administração do sistema KDD, o processo de alteração de senha do operador autenticado ou ainda, sair do sistema.

Figura A.1: Tela de autenticação do operador do sistema KDD.

Na tela inicial do sistema KDD é exibido o modelo de referência proposto por este trabalho. Cada módulo pode ser acessado por um clique.

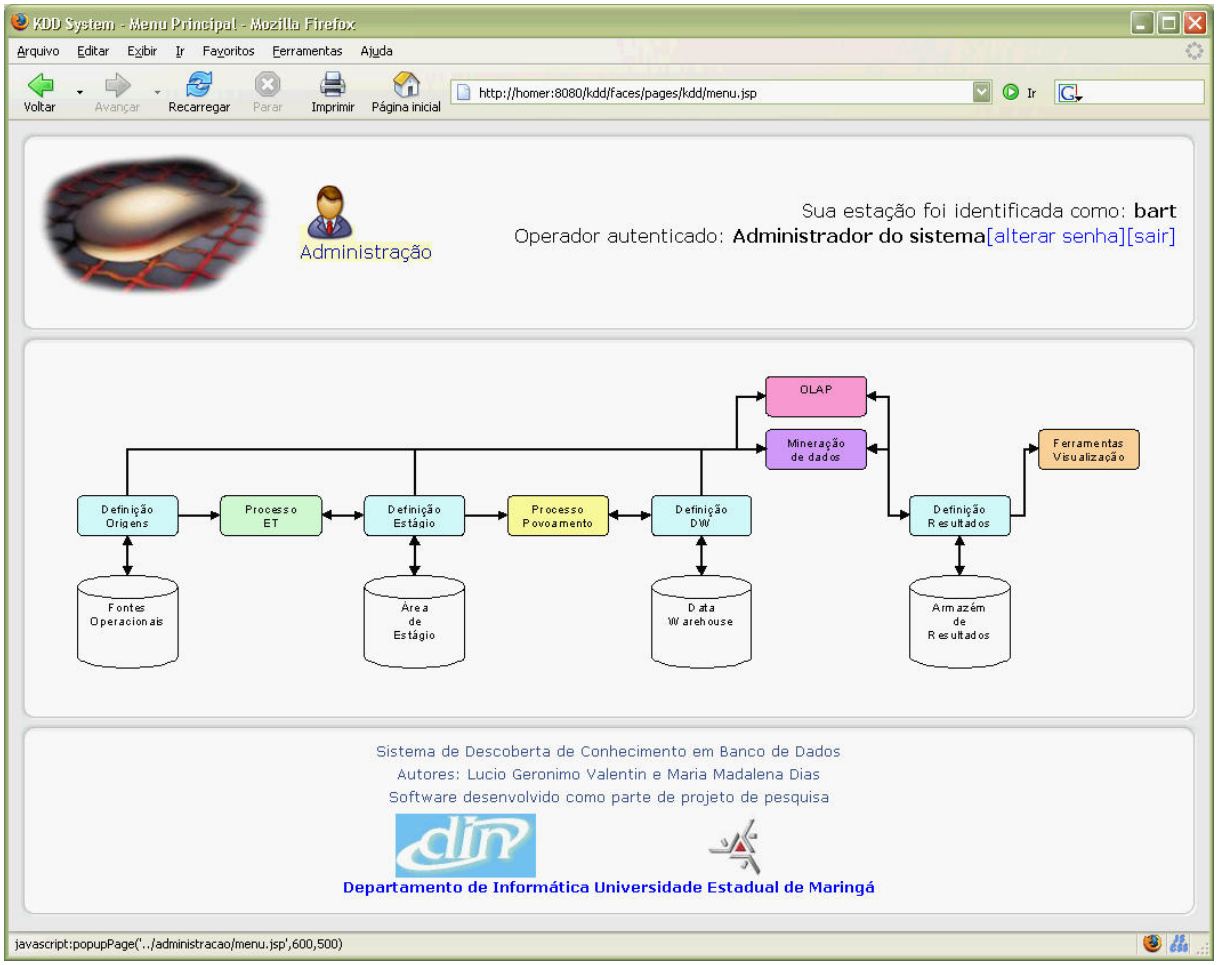


Figura A.2: Tela inicial do sistema KDD.

Administração

Acessando o ícone de Administração da tela inicial, são exibidas as opções de auditoria, controle de operadores e menu de entidades, mostradas na Figura A.3.

Na opção ‘Auditoria de alterações de cadastros e execução de processos’ é possível acessar pesquisas que mostram o que foi executado e alterado dentro do sistema KDD. A Figura A.4 mostra o resultado de uma pesquisa realizada para buscar quem e com quais parâmetros está sendo executado o processo *OlapExecute* do módulo OLAP.

No resultado da pesquisa são mostradas, as informações, sobre o processo, a identificação do operador, as datas e horas de execução, a identificação do terminal utilizado, e os parâmetros informados. A última coluna é um identificador unitário do registro de auditoria utilizado no controle interno.

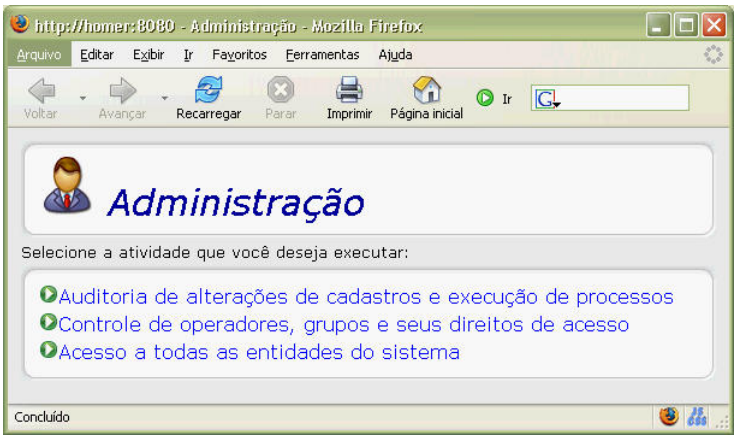


Figura A.3: Módulo de administração.

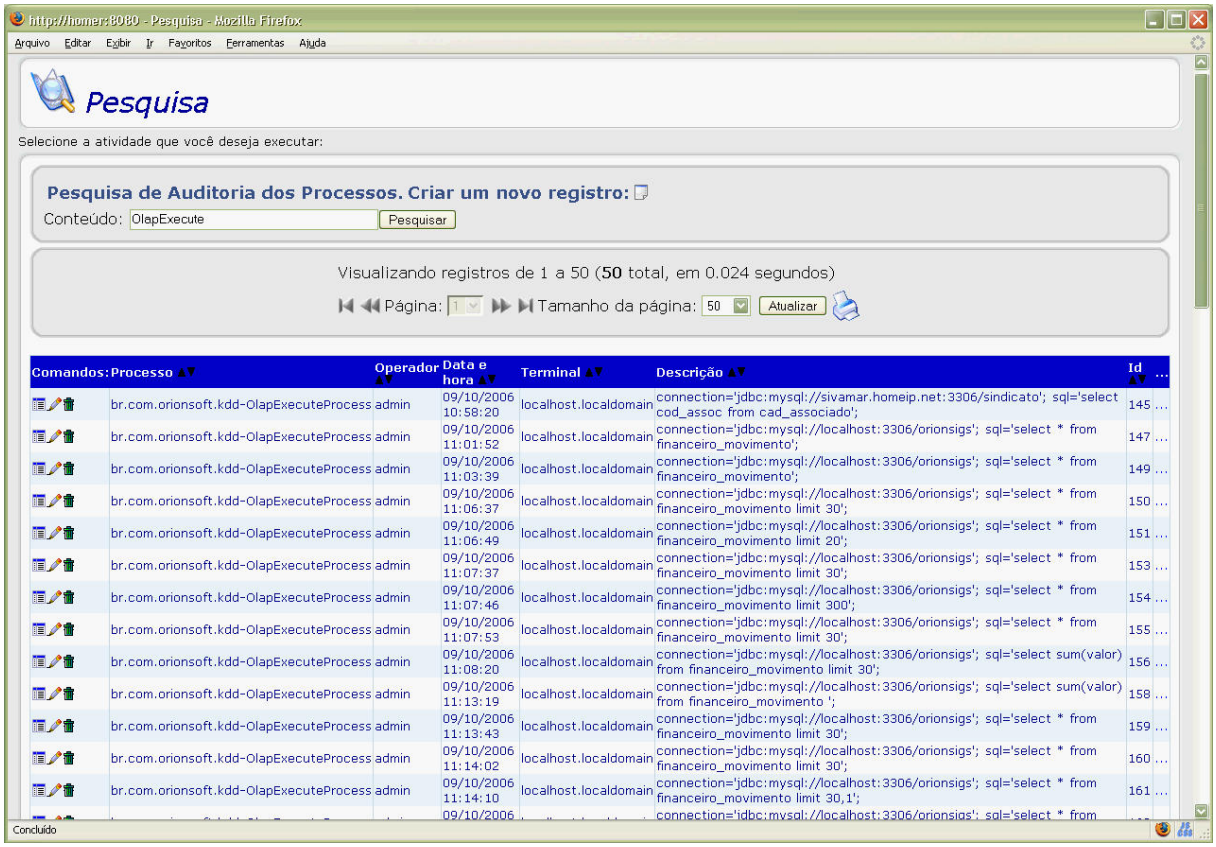


Figura A.4: Pesquisa na Auditoria de Processos que mostra a execução do processo *OlapExecuteProcess*.

Entre as opções de administração do sistema KDD, mostrado na Figura A.3, encontra-se a opção ‘Controle de operadores, grupos e seus direitos de acesso’. Através dela é possível controlar todas as estruturas de segurança do sistema KDD. A Figura A.5 mostra os processos e as entidades de segurança que foram implementados no protótipo.

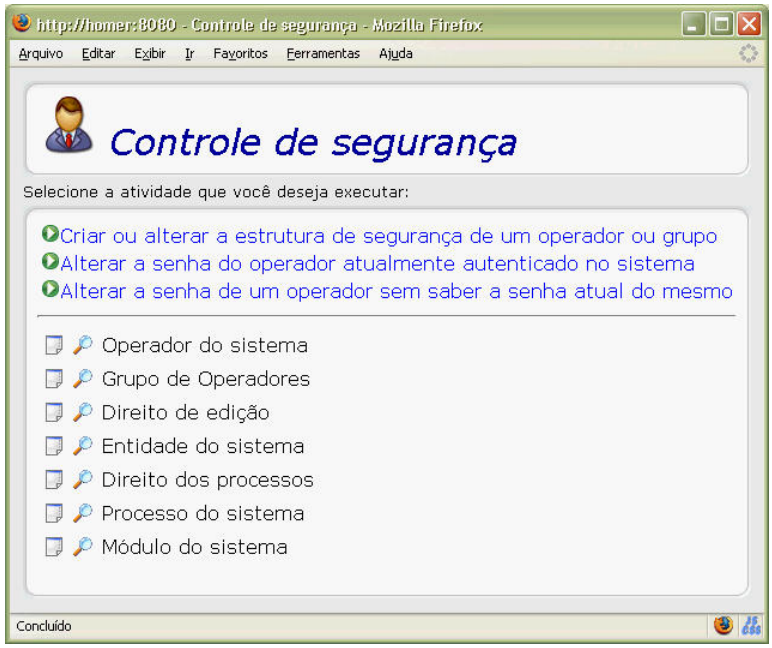


Figura A.5: Opções do controle de segurança.

A opção ‘Cria ou alterar a estrutura de segurança de um operador ou grupo’, mostrada na Figura A.5, permite que novos processos ou entidades que tenham sido integrados no sistema KDD sejam cadastrados nos direitos de segurança de um grupo automaticamente sem a necessidade de atualização dos arquivos de configuração de segurança.

A seguir são mostradas as telas de dois processos que foram implementados para a alteração de senha. O primeiro processo, ‘Alterar senha de acesso de um operador’, mostrado na Figura A.6, permite que qualquer operador autorizado possa alterar a senha de um determinado operador. Este processo é útil para que o administrador do sistema altere a senha de operadores que esqueceram ou não sabem sua senha. O segundo processo ‘Alterar senha de acesso’, mostrado na Figura A.7, permite que o operador, atualmente autenticado no sistema KDD, altere sua senha de acesso.

É interessante lembrar que a estrutura de direitos do sistema é definida por processo. Assim, é possível definir quais operadores terão direito de execução do primeiro ou do segundo processo.

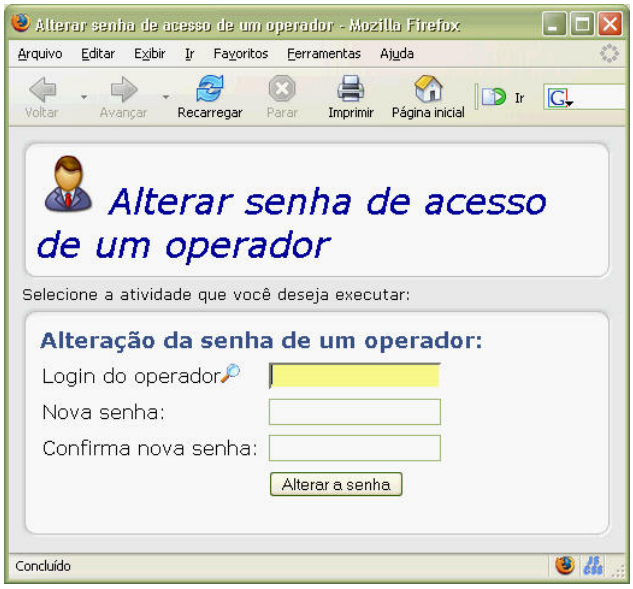


Figura A.6: Processo de alteração de senha de um operador.

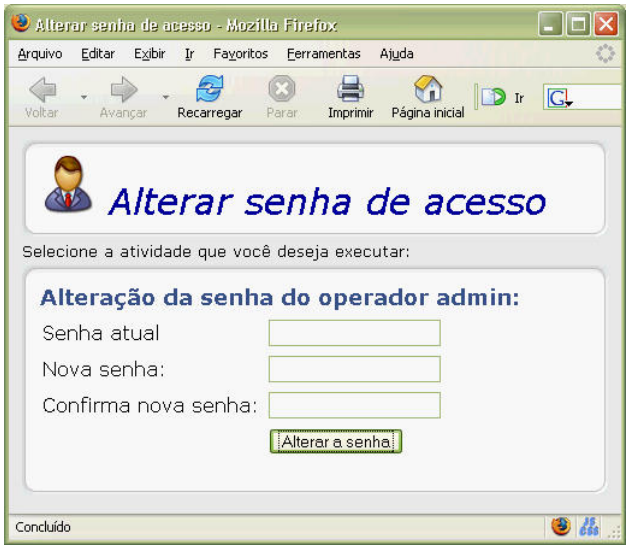


Figura A.7: Processo de alteração de senha do operador autenticado.

Definição Origens

Clicando sobre a figura do módulo ‘Definição Origens’ na tela inicial é apresentado o menu de definição das origens, mostrado na Figura A.8. Seguindo a arquitetura de software definida para o sistema KDD, foi implementado o processo *DefineSourcePropertiesProcess*, mostrado na Figura 3.11, e as três entidades utilizadas para mapear as estruturas dos bancos de dados de origem. Para cada entidade são mostradas duas opções: uma para a criação de nova instância da entidade, representada por um ícone de uma página em branco, e outra opção para pesquisa de todas as instâncias existentes, representada por um ícone de uma lupa. Os nomes

exibidos de cada entidade são obtidos nos metadados do módulo *CRUD*. Desta forma, qualquer alteração desses metadados é automaticamente refletida nas telas do sistema KDD.

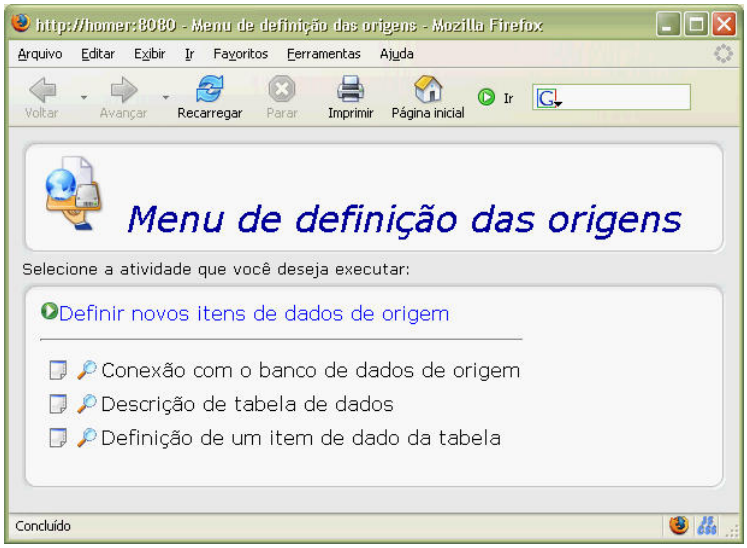


Figura A.8: Definição das bases de dados de origens.

Para a definição dos itens de dados de origem é necessário criar uma conexão que descreve os parâmetros de acesso ao gerenciador de banco de dados. Com o mecanismo *CRUD* é possível cadastrar uma nova conexão utilizando os serviços e processos genéricos para as entidades. Em outras palavras, não é necessário que o programador crie telas de cadastros específicas para cada nova entidade integrada no sistema. O gerador de visualização do mecanismo *CRUD* analisa os metadados de cada entidade e define o tipo de entrada, algumas validações e outras informações textuais necessárias durante a manutenção de uma instância de uma entidade.

A Figura A.9 mostra uma tela gerada pelo mecanismo *CRUD* para cadastrar uma nova conexão com os dados de origem. Em destaque na figura é informado que um campo de informação definido como requerido nos metadados da entidade não foi corretamente preenchido.

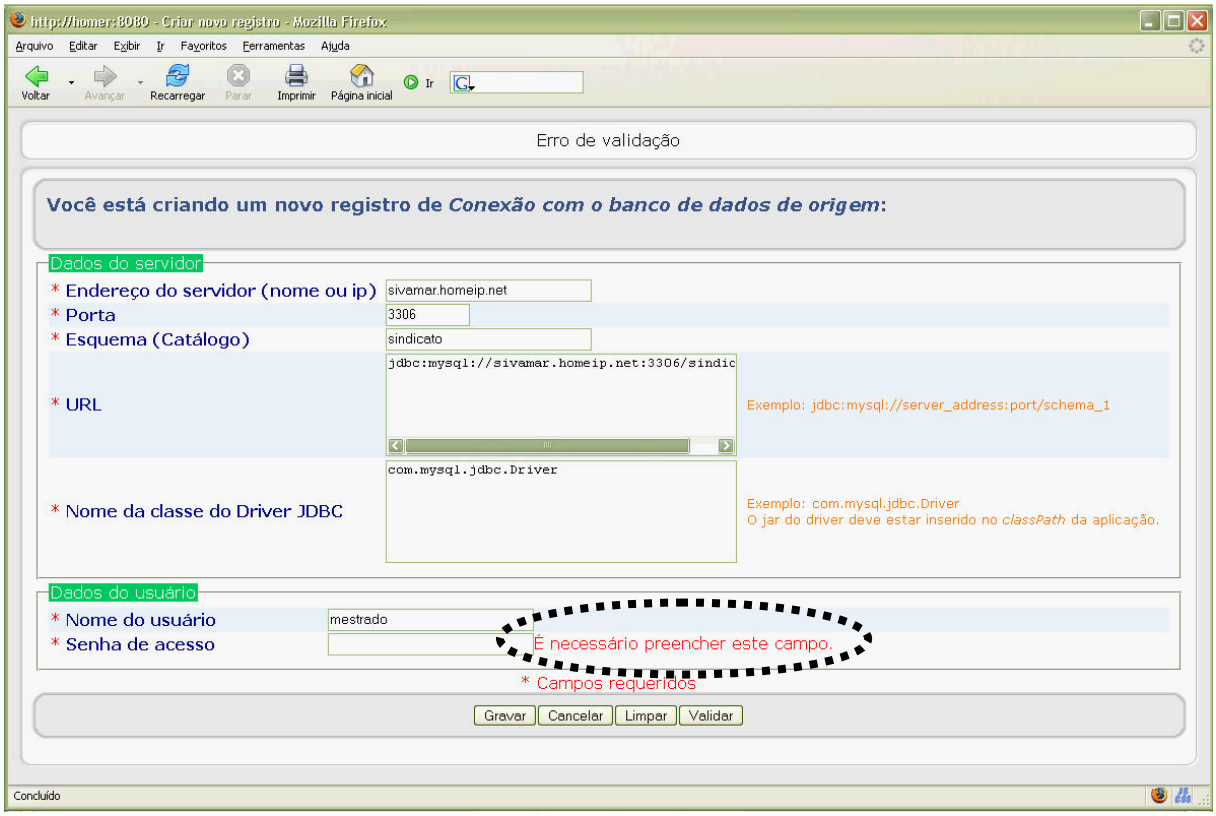
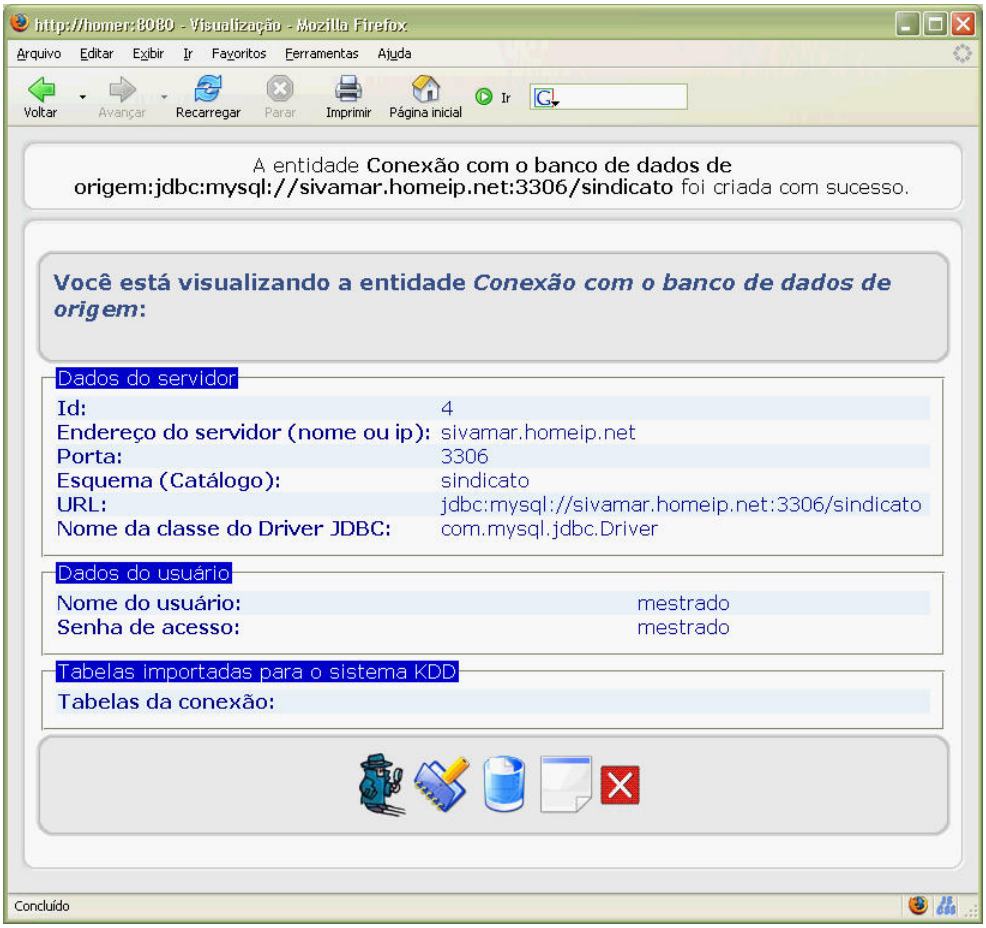


Figura A.9: Cadastro de uma nova conexão de origem. Em destaque é informado que um campo requerido não foi preenchido corretamente.

Preenchendo todas as informações necessárias, o operador pode clicar no botão ‘Gravar’ para que a nova instância da entidade seja persistida. A FiguraA.10 mostra a tela de visualização que informa que o cadastros foi realizado com sucesso e exibe as informações que foram preenchidas. Esta tela também é gerada automaticamente pelo mecanismo *CRUD* baseando-se nos metadados da entidade.

O ícone de um detetive permite que o operador acesse uma tela com o relatório de quem criou ou alterou a entidade visualizada. Este recurso é a implementação do serviço *CheckAuditCrudService* mostrado na Figura 4.14. O relatório gerado é mostrado na Figura A.11.



FiguraA.10: Tela de visualização dos dados recentemente cadastrados.

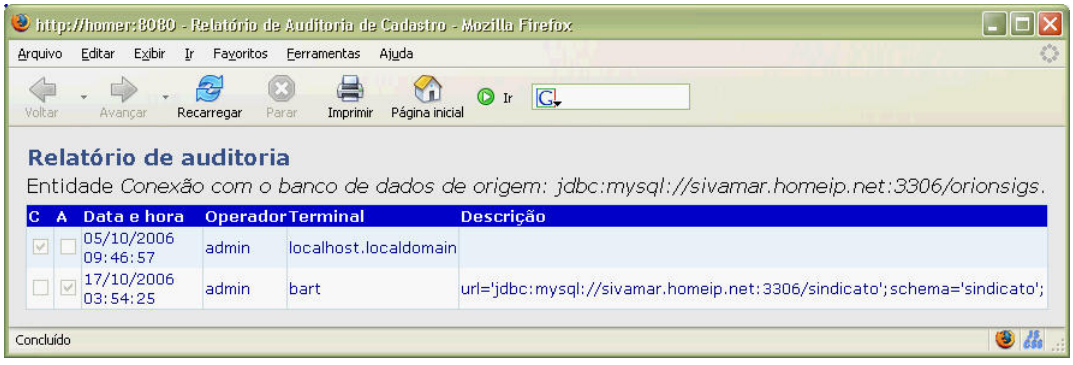


Figura A.11: Relatório de auditoria de uma entidade do sistema.

A seguir são mostradas as telas que foram criadas para os passos do processo *DefineSourcePropertiesProcess*. Este processo é acessado no menu da Figura A.8 pela opção 'Definir novos itens de dados de origem'. A implementação deste processo segue a especificação do caso de uso 'Definir Origens' mostrado na Figura 3.10.

O primeiro passo deste processo é a escolha da conexão de origem que será utilizada. A Figura A.12 mostra a escolha da conexão com recursos para criar uma nova conexão se for

necessária (ícone de uma página em branco), opção de atualização da lista de conexões disponíveis (ícone de flechas circulares) e opção de abrir uma tela de pesquisa das conexões disponíveis para realizar alguma manutenção nos cadastros caso seja necessária (ícone de uma lupa).

Figura A.12: Definição das origens - Passo 1: Escolha da conexão.

Como descrito no caso de uso, após a escolha da conexão, o sistema KDD se conecta com o gerenciador de banco de dados definido e obtém a lista de tabelas existentes. Esta lista é exibida para o operador, como mostra a Figura A.13, e possibilita a escolha da tabela de origem que será analisada. Passando para a próxima etapa, o sistema KDD obtém a lista de itens de dados disponíveis na tabela selecionada, mostrados na Figura A.14, e permite que o operador selecione quais itens de dados ficarão disponíveis no sistema KDD para os demais processos. Clicando no botão ‘Confirmar Seleção’ o operador receberá uma mensagem informando que sua seleção foi persistida com sucesso. Esta mensagem é mostrada na Figura A.15.

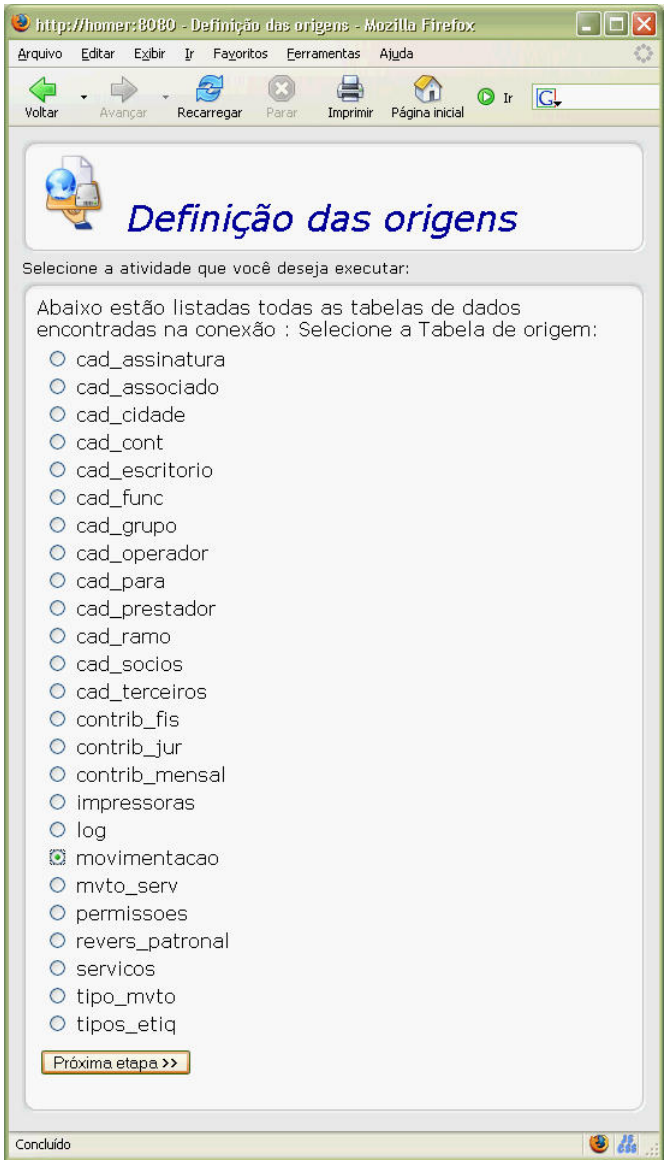


Figura A.13: Definição das origens - Passo 2: Escolha da tabela de origem.

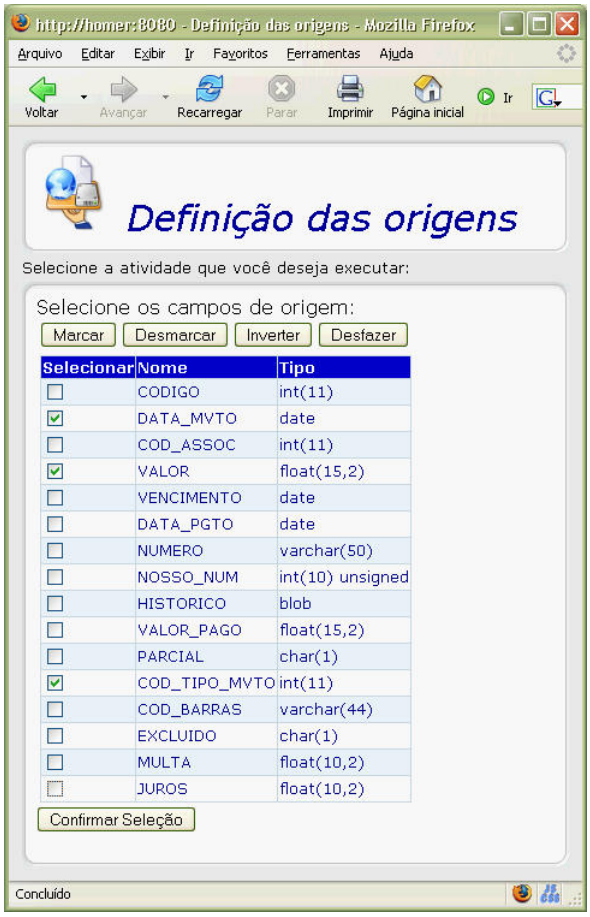


Figura A.14: Definição das origens - Passo 3: Escolha dos itens de dado.

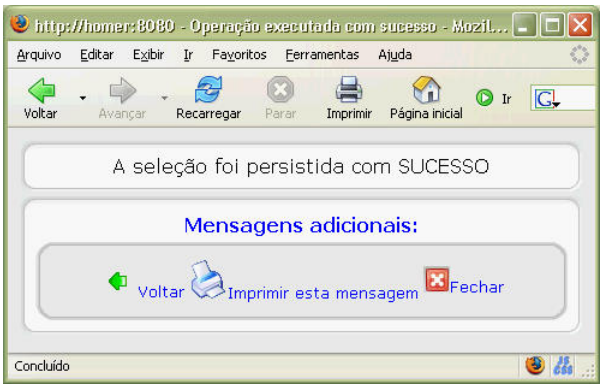


Figura A.15: Definição das origens - Passo 4: Mensagem de seleção efetuada com sucesso.

Concluída as definições dos itens de dados de origem, o próximo passo é a definição da Área de Estágio.

Definição Estágio

Seguindo o mesmo padrão de menu e de telas da seção anterior, a Figura A.16 mostra o menu de definição da Área de Estágio.

A definição da origem apresentada na seção anterior tem o objetivo de acessar a origem e mapear alguns itens de dados para o sistema KDD, isto porque, a origem já existe independente do sistema KDD. Já a Área de Estágio é totalmente definida dentro do sistema KDD. Para que as bases de dados da Área de Estágio fiquem consistentes com as estruturas definidas dentro do sistema KDD (Conexão da Área de Estágio, Descrição de tabela de dados da Área de Estágio, Definição de um item de dado da tabela da Área de Estágio) foi implementado o processo 'Sincronizar as definições da Área de Estágio' que verifica quais tabelas e itens de dados foram alterados, incluídos ou excluídos das estruturas do sistema KDD.

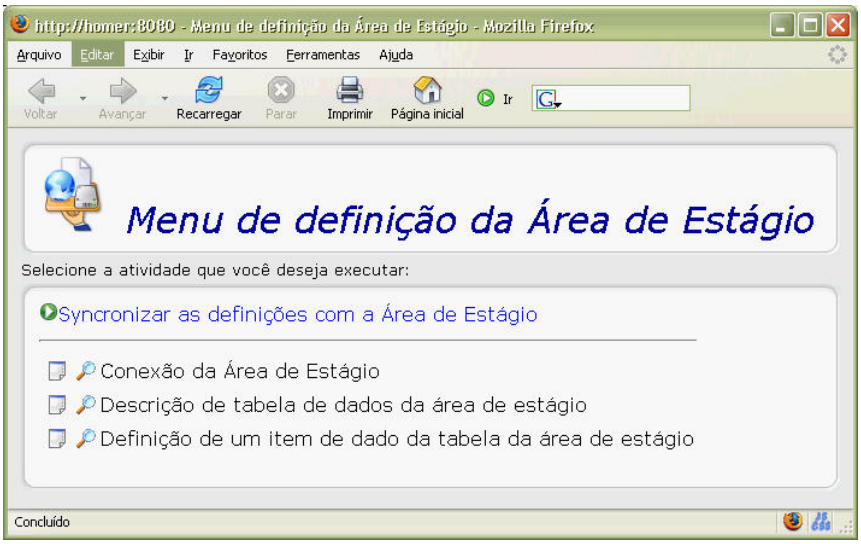


Figura A.16: Definição da Área de Estágio.

O processo de sincronização possui somente um passo, mostrado na Figura A.17, para a seleção da conexão da Área de Estágio que será sincronizada. Clicando no botão 'Sincronizar' o sistema KDD dispara o serviço de sincronização.

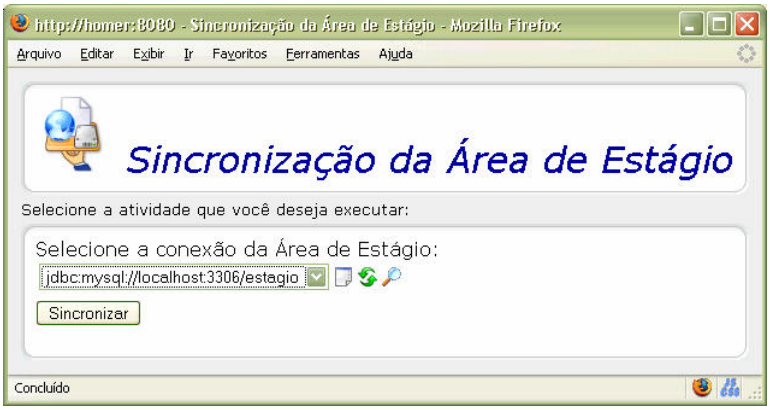


Figura A.17: Processo de sincronização da Área de Estágio.

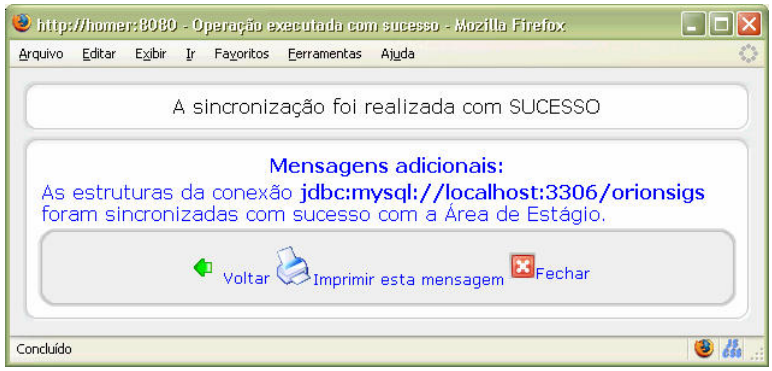


Figura A.18: Mensagem de sucesso da sincronização da Área de Estágio.

Extração e Transformação

Para que uma extração seja executada é necessário definir todos os seus parâmetros preenchendo uma estrutura chamada ‘Definição de estagiamento’ que é a implementação da entidade *EtDefinition* mostrada na Figura 3.12. A Figura A.19 mostra o menu de extração e transformação que permite preencher a estrutura de *estagiamento* e acessar o processo de execução das extrações definidas.

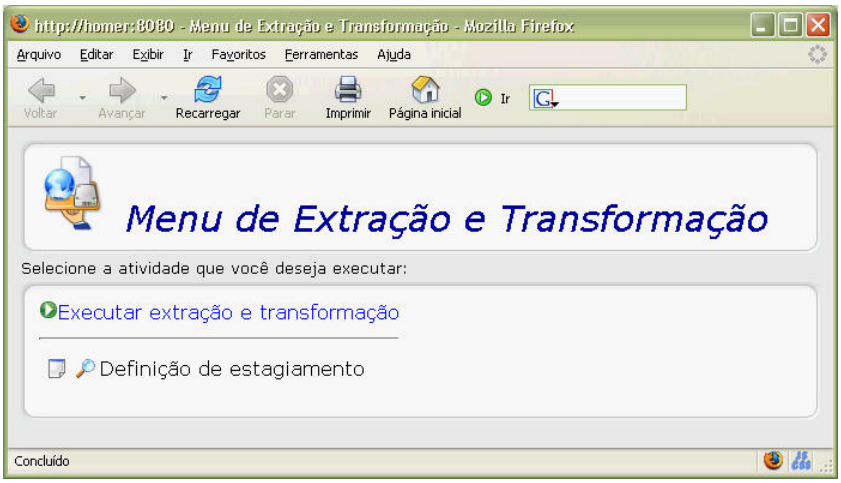


Figura A.19: Extração e transformação dos dados.

A Figura A.20 mostra a tela de preenchimento dos dados de uma definição de *estagiamento*. Figura A.21 mostra a tela que é exibida após o preenchimento ser concluído. O ponteiro apontado sobre a propriedade ‘Item de dado de destino’ destaca o recurso de *links* que permite navegar entre as entidades relacionadas.

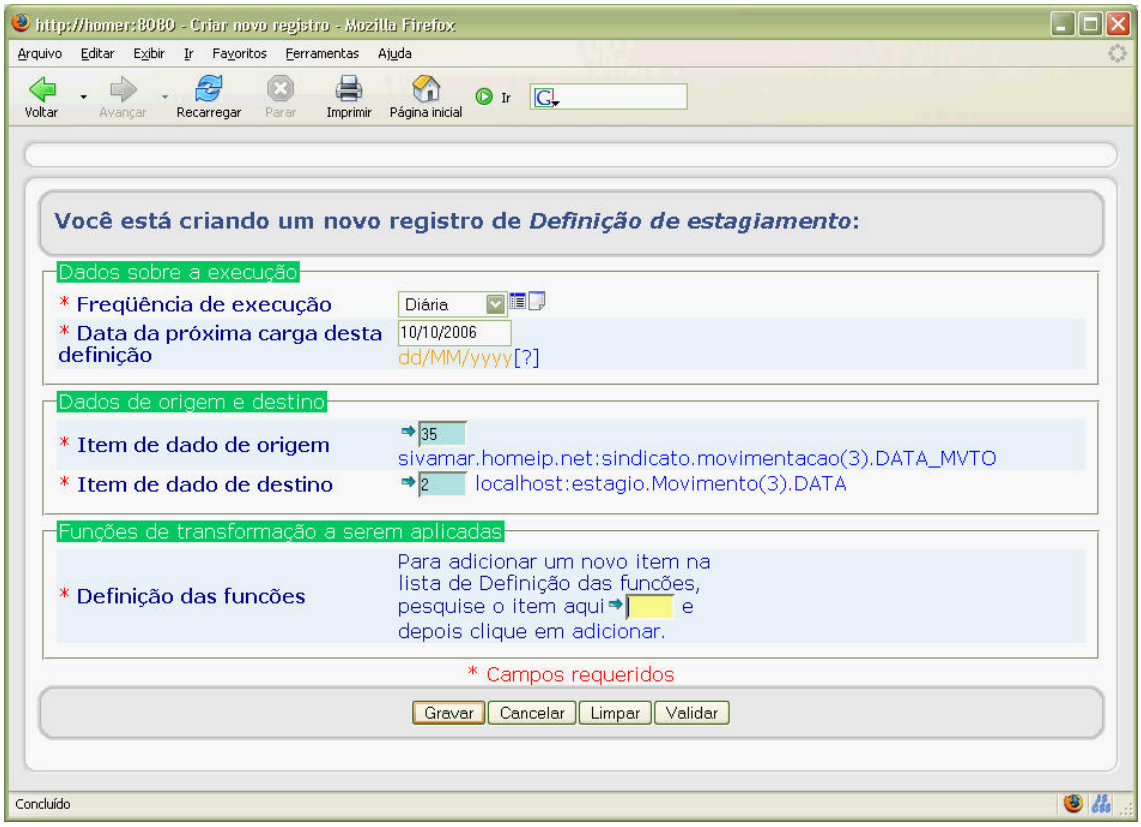


Figura A.20: Criação de uma nova definição de *estagiamento*.

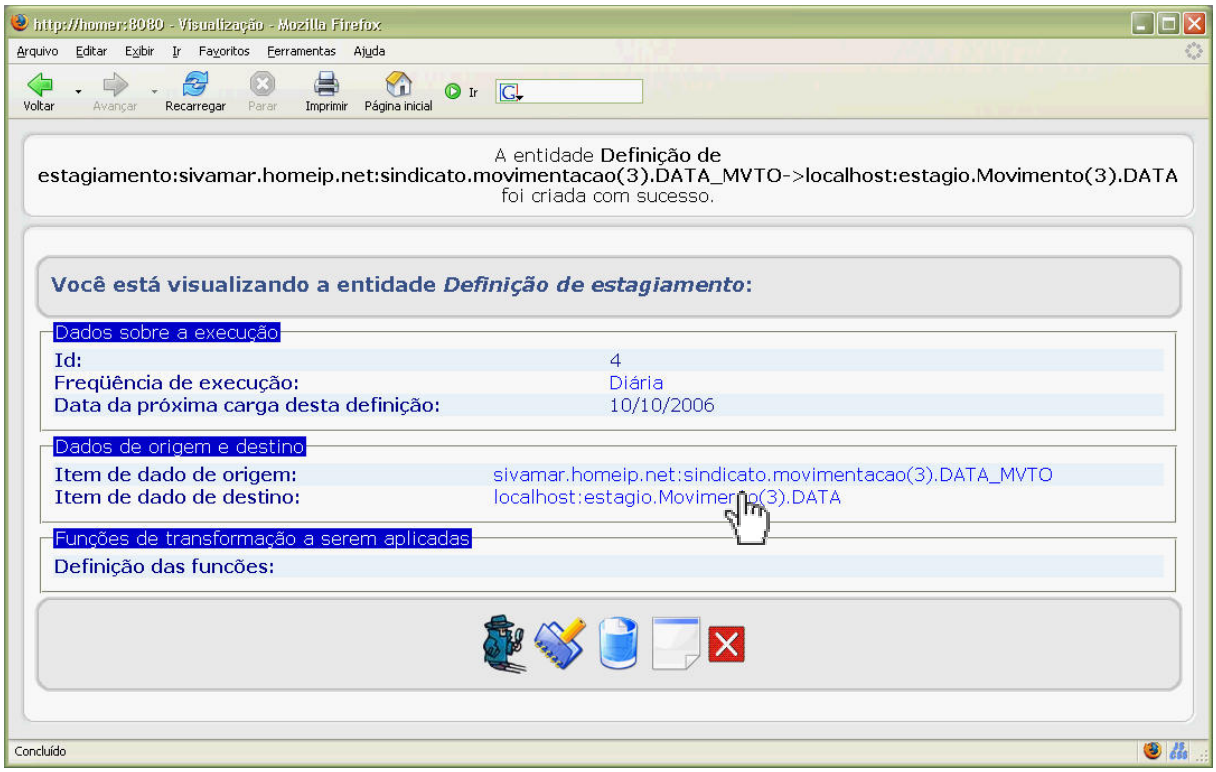


Figura A.21: Visualização dos dados cadastrados da nova definição de *estagiamento*.

Com as dados de *estagiamento* preenchidos, é possível acionar o processo de execução de extração e transformação que seleciona quais definições estão disponíveis para execução em um determinado período. Vale lembrar que cada definição possui uma propriedade 'Freqüência de execução' e 'Data da próxima carga...', mostradas na Figura A.20, que permitem controlar a disponibilidade das extrações e transformações por período.

A Figura A.22 mostra a seleção de definições de *estagiamento* disponíveis para execução. É possível, neste momento, marcar quais definições devem ser realmente executadas. Após a execução, o operador recebe uma mensagem indicando que as extrações e transformações foram executadas com sucesso.

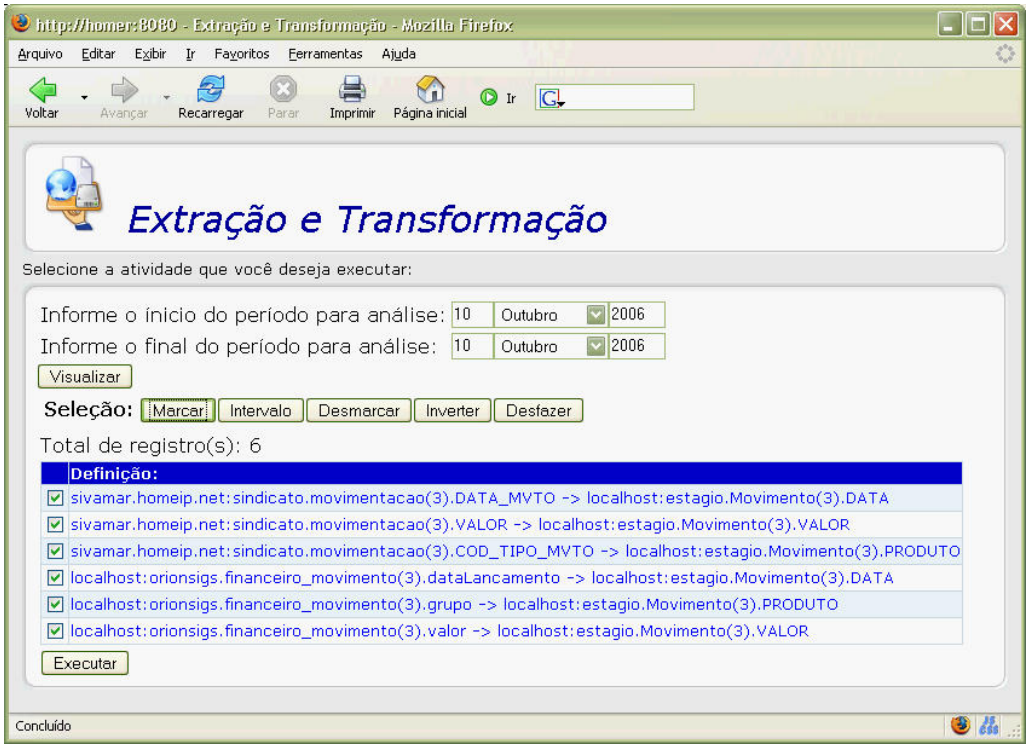


Figura A.22: Processo de execução da extração e transformação dos dados.

Definição do DW

A definição do DW segue o mesmo padrão de telas e estruturas já apresentadas.

A Figura A.23 mostra o menu de definição do DW. Neste menu é possível definir as conexões do DW, as dimensões, os fatos e suas propriedades.

A Figura A.24 mostra os dados de uma conexão do DW. Os fatos e dimensões definidos para esta conexão.

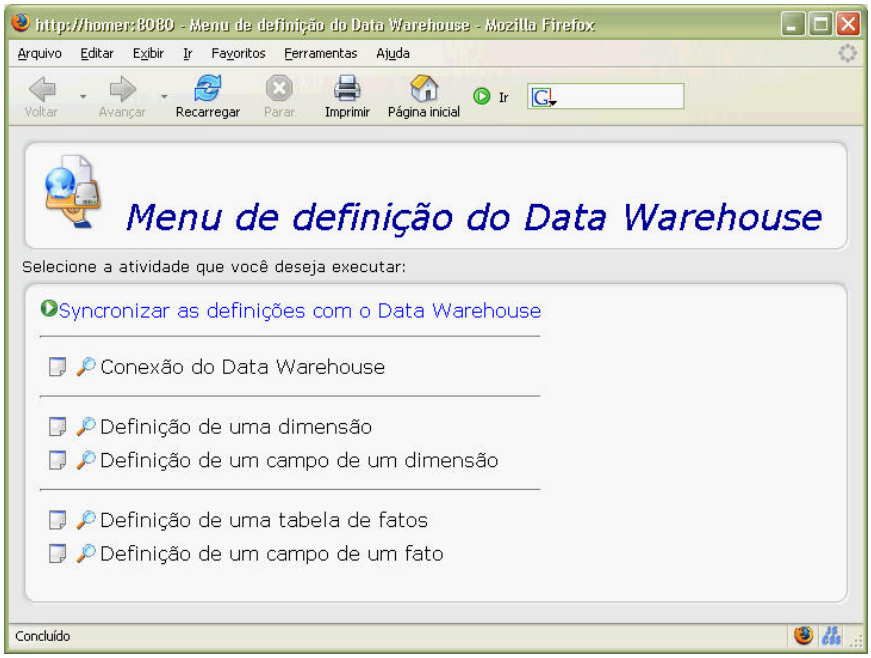


Figura A.23: Definição das estruturas do DW.

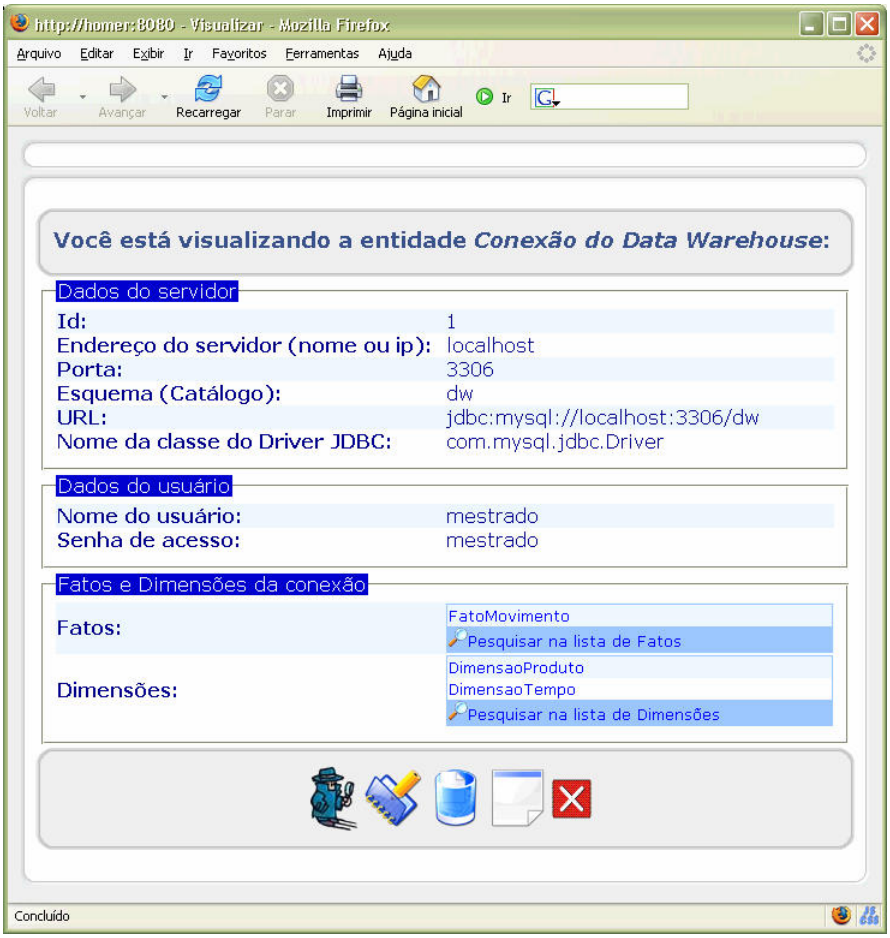


Figura A.24: Visualização dos dados de uma conexão do DW.

Povoamento

A definição do povoamento segue o mesmo padrão da definição de ET. Existe um processo de execução de povoamento e uma estrutura com os dados dos povoamentos que devem ser executados. A Figura A.25 mostra a tela de menu de povoamento do DW.

Figura A.25: Povoamento do DW.

OLAP

Como proposto no modelo de referência, as operações de OLAP têm como fonte de dados as bases de origem, a área de estágio, o DW e o armazém de resultados.

O primeiro passo para a execução de uma operação de OLAP é a escolha de uma conexão. A Figura A.26 mostra a listagem das conexões disponíveis no sistema KDD.

Depois que uma conexão é selecionada, o operador poderá entrar com algumas instruções SQL para realizar suas consultas. A Figura A.26 mostra a execução de uma consulta na base de dados de origem utilizada no estudo de caso deste trabalho. Nesta consulta são selecionadas as pessoas cadastradas no período de 01/01/2006 a 31/06/2006. Uma junção com a tabela de cadastro de cidades é definida para que o nome da cidade seja projetado no resultado. Logo abaixo da entrada da instrução SQL o resultado da consulta é exibido.

Para salvar este resultado no armazém de resultados basta clicar no *link* ‘Salvar resultados na área de resultados’. A Figura A.27 mostra os parâmetros que devem ser fornecidos para que os resultados sejam armazenados. A Figura A.28 mostra a mensagem de sucesso do armazenamento dos resultados. A partir deste momento, os resultados da consulta podem ser visualizados diretamente do armazém de resultados.

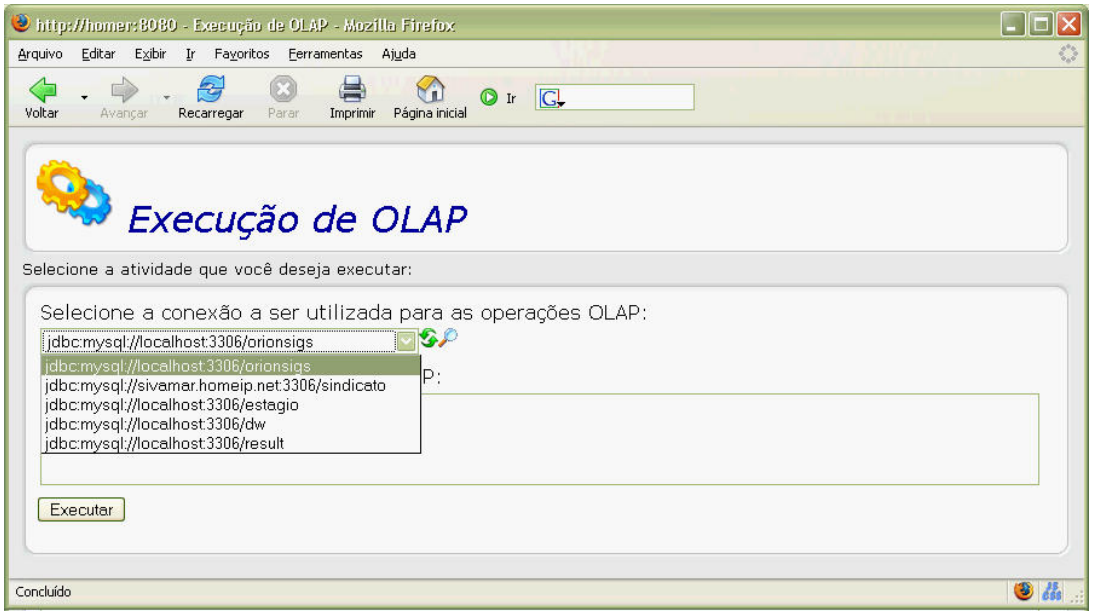


Figura A.26: Execução de operações de OLAP: Seleção da origem dos dados.

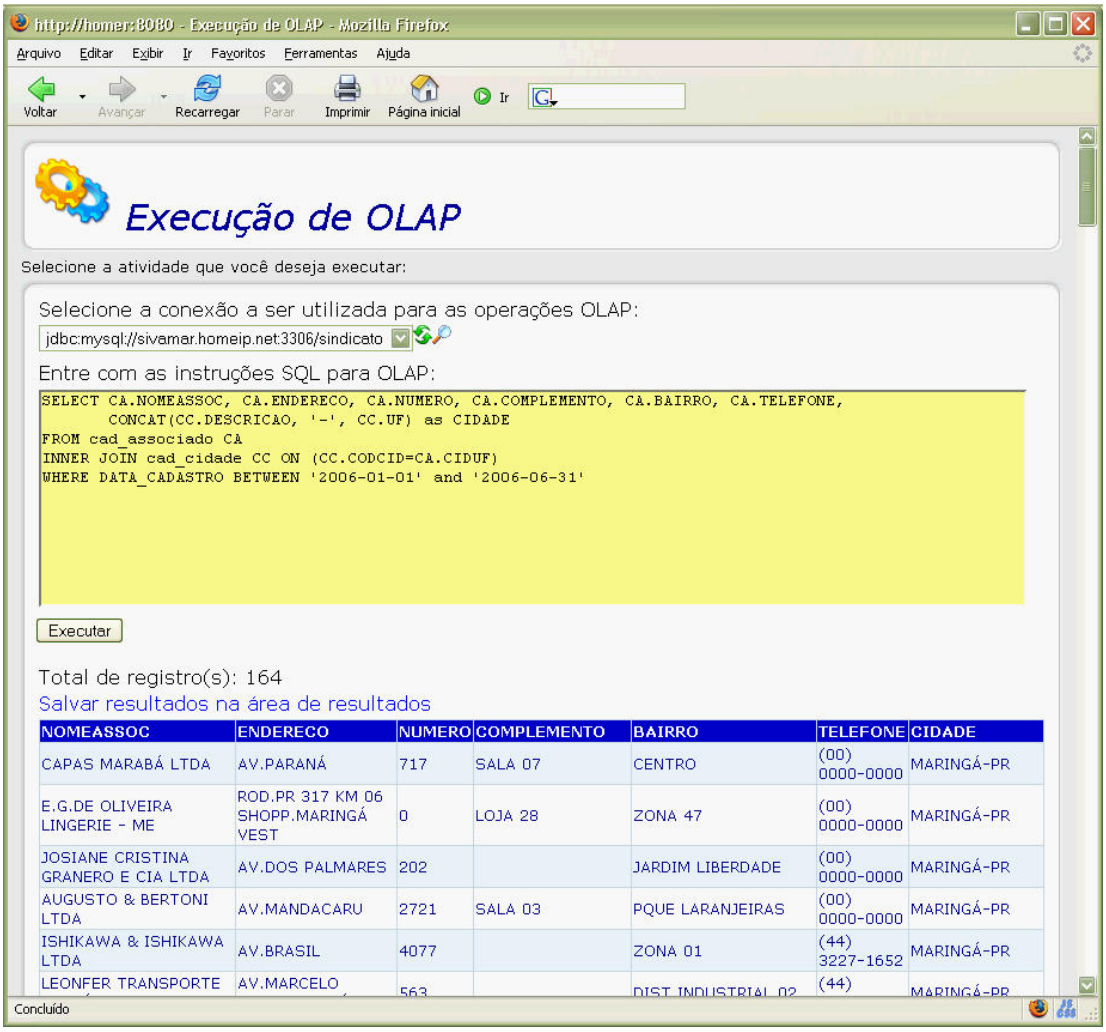


Figura A.27: Tela de execução de operações OLAP com os resultados.

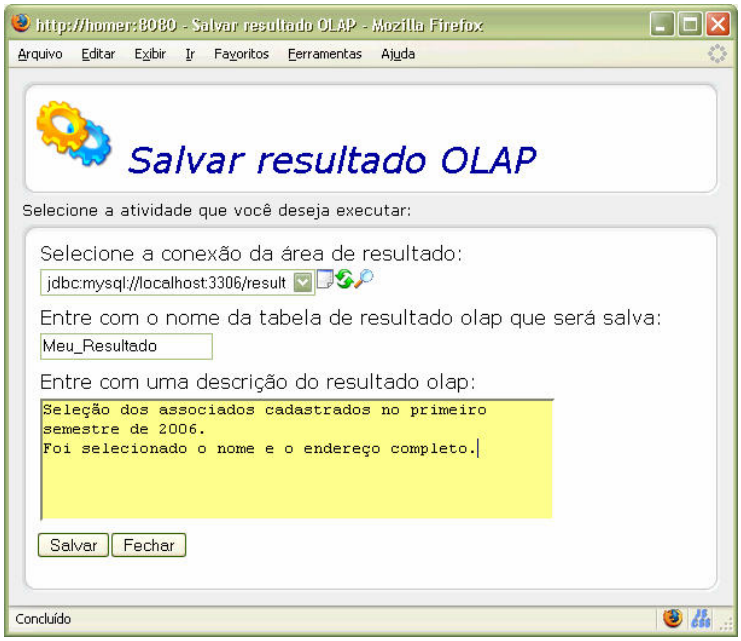


Figura A.28: Tela para armazenar o resultado de OLAP.

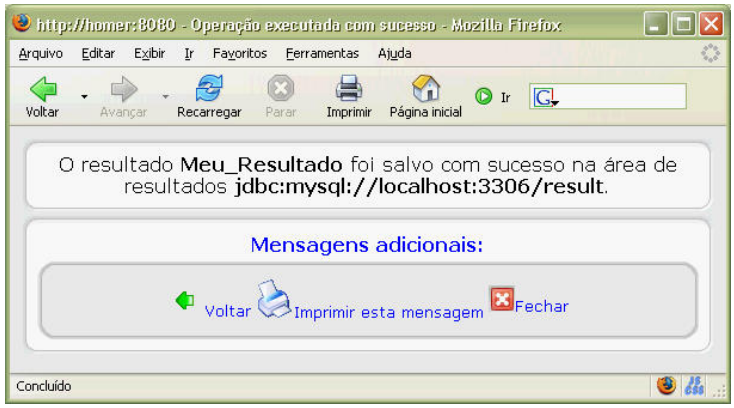


Figura A.29: Mensagem de sucesso da gravação do resultado de OLAP no Armazém de Resultados.

A Figura A.30 mostra uma consulta OLAP sendo realizada no armazém de resultados. Os resultados consultados foram gravados nos passos anteriores. Desta forma, novos resultados podem ser obtidos a partir de resultados já obtidos por outras consultas.

Figura A.30: Pesquisa no armazém de resultados para confirmar os resultados armazenados.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo