UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

AMGraA: Uma Abordagem para Migração Gradativa de Aplicações Legadas

VALDIRENE FONTANETTE

São Carlos - SP
Maio de 2005.

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

F679am

Fontanette, Valdirene.
AMGra: uma abordagem para migração gradativa de aplicações legadas / Valdirene Fontanette. -- São Carlos : UFSCar, 2005.
144 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2005.

1. Engenharia de software. 2. Reengenharia orientada a objetos. 3. Sistemas de transformação de software. 4. Integração de sistemas. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"AMGraA: Uma Abordagem para Migração Gradativa de Aplicações Legadas"

VALDIRENE FONTANETTE

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador – DC/UFSCar)

Profa. Dra. Regina Maria Maciel Braga
(UFJF)

Profa. Dra. Rosana Teresinha Vaccare Braga
(ICMC/USP)

São Carlos
Maio/2005

A DEUS, pois tudo deve ser feito no nome dELE e para ELE;

A meus pais e minha irmã, pela compreensão, apoio e carinho.

Se você acha que pode, tem razão, você pode.
Se você acha que não pode tem razão você não pode.
Henry Ford

Agradecimentos

Ao longo deste período diversas pessoas contribuíram direta ou indiretamente para a realização de mais esta conquista tão almejada.

Primeiramente, agradeço a Deus por me proporcionar esta rica oportunidade. Obrigada Senhor porque escolhestes nestes momentos tão difíceis fazer parte de minha vida. Hoje tenho segurança do teu amor e a certeza de que estás comigo.

Agradeço ao meu professor e amigo XUXA por todo incentivo e apoio para tentar o mestrado, me apresentando a universidade e os professores.

Em seguida, agradeço ao meu orientador Prof. Dr. Antonio Francisco do Prado, pela oportunidade concedida, pela orientação e confiança depositada em mim, e pelas inúmeras discussões sobre pontos de vista diferentes, mas que me ajudaram muito a engrandecer como pesquisadora.

Ao André Luis Costa de Oliveira, pela grandiosa oportunidade de montar um projeto em conjunto com uma empresa, pelas inúmeras conversas, orientação, incentivo e paciência dispensada a mim.

À minha família, pelo apoio sempre: ao meu pai Valdir pelas palavras de incentivo e encorajamento, mesmo não conhecendo sobre os assuntos pertinentes a um projeto de mestrado; à minha mãe Marta, pela sua preocupação e seu "colinho" não importando a hora; à minha irmã Amábile, pela amizade e compreensão nos momentos difíceis. Vocês são muito importantes para mim.

Aos meus tios Dorival e Ana Amélia, por incentivarem em mais esta conquista dentre tantas outras já me proporcionadas.

À minha amiga Márcia, por me acolher tão bem em São Carlos e pela amizade até hoje.

Ao Calebe, pelo seu carinho e paciência, enquanto estudávamos juntos tantas e tantas vezes para LFA e pelos ótimos momentos que passamos juntos. Você nunca será esquecido.

Ao André, meu amigo de todas as horas, nem sempre horas tão divertidas, né Andrezitcho? Obrigada pela sua atenção e carinho comigo, pelas madrugadas em que fizemos companhia um para o outro. Você é muito querido. Espero continuar a honrar esta bela amizade de onde estiver.

Ao Reinaldo, uma das primeiras pessoas que conheci aqui e que se tornou tão presente e importante pra mim. Querido Rei, obrigada por estar ao meu lado nestes períodos tão difíceis que passei sempre disposto a me ouvir e me consolar com suas palavras. Te admiro muito, não só sua pessoa, mas o profissional que você está se tornando. Quando eu crescer quero ser igual a você, viu?!?

Às minhas amigas de república, Ariane e Mariana, pelos momentos de amizade e descontração. Obrigada pela atenção e preocupação quando eu não chegava em casa no horário do último bus. Pelos momentos de descontração na procura da nossa nova casa e até mesmo pelas intriguinhas (Hehehe).

Às mais novas companheiras de república, Amanda e Luanna, pelo carinho e atenção, mesmo nos conhecendo tão pouco. Saibam que podem contar comigo sempre. Espero que vocês tenham a honra de fazer amigos tão lindos como estes que eu fiz.

Aos companheiros de pesquisa, Vinicius, Joãozinho, Daniel e Raphael pelos momentos que passamos juntos no laboratório.

Aos parceiros que colaboraram para realização do projeto PIPE, Edimilson, Celso, Ariele, Marcos e Nicola.

Aos companheiros de outras épocas, Adail, Diogo, Balbino, Eduardo, Bossonaro, Moraes, Clo e Darley.

Aos colegas de pós-graduação, Jéssica, Joelle, Vânia, Fabiana, Eloíze, Renatinho e Américo.

Aos professores do departamento, em especial Sandra Abbib (Sandroca) por me ouvir e aconselhar sempre, Zorzo pela orientação na disciplina de PESCD e ao Mauro pelo apoio e pelas sugestões na qualificação.

Aos funcionários do departamento, em especial Cristina, Luzia, Ofélia e Ana.

À CAPES e FAPESP, pelo apoio financeiro.

E, finalmente, às demais pessoas que, direta ou indiretamente, estiveram desenvolvidas para a realização desse trabalho.

Meu sincero MUITO OBRIGADA!

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Ao longo do tempo, as aplicações normalmente precisam evoluir para acompanhar e atender às novas tecnologias que surgem ou amadurecem nas diferentes áreas da computação, como ocorre, por exemplo, com as linguagens de programação. A partir dessa exigência, novas pesquisas estão sendo realizadas para oferecer suporte a essa evolução contínua. Esse quadro aprimorou-se com o amadurecimento das tecnologias para *Web*, uma vez que a necessidade da empresa hoje nem sempre é reconstruir totalmente uma aplicação, mas adaptá-la para acomodar essas novas tecnologias. A modernização de aplicações antigas, denominadas legadas, é uma tarefa árdua e de alto custo para as empresas. Embora existam na literatura várias abordagens que apóiam este processo, ainda há uma carência de recursos que ofereçam flexibilidade, aproveitem investimentos anteriores das empresas e consigam resultados práticos. Deste modo, o processo de modernização nas empresas é realizado quase sempre de forma manual e *ad-hoc*. Motivado por estas idéias, este trabalho apresenta a AMGraA, uma abordagem para a modernização gradativa de aplicações legadas, reconstruindo-as para atender às novas tecnologias que surgiram ao longo do tempo. Este projeto de pesquisa resultou de uma parceria entre o GOES (Grupo de Engenharia de *Software*), uma empresa de desenvolvimento de *software*, e a Fapesp, em apoio ao programa de desenvolvimento à pesquisa em pequenas empresas.

Abstract

Along the time the applications usually need to evolve to catch up with and to assist to the new technologies that appear or get mature in the different computing areas, as it happens, for instance, with the programming languages. Starting from that demand, new researches are being accomplished to offer support to this development. This situation was improved with the ripening of Web technologies, once the need of the company nowadays is not always to convert an application totally, but to adapt these applications to accommodate these new technologies. The modernization of old applications, denominated legacy applications, is an arduous and a high cost task for the companies. Although there are several approaches that support this process in the literature, there is still a lack of resources that offer flexibility, take advantage of previous investments of the companies and get practical results. This way, the modernization process in the companies is almost always accomplished in a manual and *"ad-hoc"* form. Based on these ideas, this work presents AMGraA, an approach for gradual modernization of legacy applications, reconstructing them to attend to new technologies that came out along the time. This research project resulted of a partnership among GOES (Group of Software Engineering), a software development company, and FAPESP in support to the research and the development in the small companies.

Capítulo 1

Introdução

1.1 Contextualização

Uma aplicação de *software* é um artefato evolutivo e, com o decorrer do tempo, seu projeto e implementação originais precisam ser modificados para atender a novos requisitos e/ou melhorar o seu desempenho, incorporando conhecimentos substanciais do seu contexto. A este processo dá-se o nome de "Manutenção de *Software*". Mas, depois de certo tempo, as aplicações tornam-se obsoletas, pois a tecnologia em que as mesmas foram desenvolvidas torna-se antiga e, na maioria das vezes, não têm mais o suporte de seus fabricantes, o que requer sua modernização. A essas aplicações, dá-se o nome de aplicações legadas. Nestes casos, pode-se conduzir a *"Modernização de Software"* [SEACORD et al. 2003] que envolve mudanças mais extensas que a manutenção, porém, quase sempre conservando uma porção da aplicação existente. Ela envolve a reestruturação da aplicação e a inserção de novas funcionalidades, onde novas tecnologias e novas práticas são aplicadas.

A evolução dos negócios de uma empresa ao longo dos anos requer uma evolução sincronizada de suas aplicações legadas, porém, tais aplicações deveriam sempre oferecer um nível de qualidade adequado, de forma que estas pudessem ser mantidas facilmente. Infelizmente, devido às suas desatualizações, as aplicações legadas muito freqüentemente possuem baixos níveis de qualidade, documentação escassa ou desatualizada, e, como conseqüência, suas manutenções ficam muito caras [BIANCHI et al. 2003].

Essas aplicações são normalmente essenciais para o bom funcionamento das empresas ou até mesmo críticas. A reconstrução dessas aplicações usando novas tecnologias é uma boa solução, ao invés de construir uma nova aplicação do início, o que geralmente envolve riscos e um processo complexo e trabalhoso.

Há várias abordagens e técnicas na literatura, que apóiam o processo de modernização de aplicações, visando decompor aplicações, manipular, analisar [SYSTA 1999], sintetizar [BIGGERSTAFF et al. 1994], componentizar [ALVARO et al. 2003] e visualizar artefatos de *software* [PRICE et al. 1993]. Entretanto, estas abordagens e ferramentas ainda são complexas para o uso amplo na indústria, pois exigem alto conhecimento técnico e possuem algumas limitações, como a falta de flexibilidade no processo de migração e de integração com outras ferramentas, fazendo com que a empresa ao adotá-las perca investimentos realizados anteriormente.

Deste modo, o processo de modernização é realizado quase sempre de forma manual e ad-hoc pelas empresas, onde os Engenheiros de *Software*, que conhecem a aplicação legada, reescrevem-na em uma nova tecnologia, incorporando recursos atuais.

As propostas de modernização realizadas de forma manual muitas vezes desencorajam as empresas ou até mesmo inviabilizam sua execução devido aos altos custos e a necessidade de realocar pessoas que realizam a manutenção para trabalharem no desenvolvimento da nova aplicação. A modernização manual também é muitas vezes inviabilizada pelo risco da reescrita de uma aplicação totalmente nova, pois o novo código pode não ter as mesmas funcionalidades (o código legado pode ter rotinas complexas não expostas, de difícil entendimento e análise ou uma série de manutenções sobre manutenções, modificando várias vezes as mesmas informações) ou até mesmo ser finalizado com muitos erros devido à complexidade e falta de experiência da equipe nas novas tecnologias.

A migração de uma aplicação legada pode ser feita de várias formas, porém, todas exigem diferentes mecanismos e controles. É por isto que muitas ferramentas não conseguem atender a este mercado. Um dos requisitos é que a aplicação legada precisa ser migrada aos poucos e ao mesmo tempo precisa continuar funcionando. Desta forma, em muitos casos é necessário adicionar novas "camadas" na aplicação para que haja integração entre a nova camada construída e a aplicação legada. Assim, a substituição é feita aos poucos, o resultado é mais rápido, e o usuário da aplicação não sente tanto os erros envolvidos em uma migração, e que causam problemas, acarretando um longo tempo de espera, devido à má compreensão do legado e à falta de funcionalidades.

Além dos problemas citados, as abordagens de migração não-gradativas não apresentam resultados práticos e rápidos para os usuários finais, desmotivando e fazendo com que projetos de modernização sejam cancelados devido às mudanças de prioridade na empresa.

Motivados em solucionar estes problemas e pela necessidade que as empresas têm de migrar suas aplicações de forma gradativa e com o suporte de ferramentas integradas que auxiliem no

processo de migração, o GOES (Grupo de Engenharia de *Software*)[1], a Apyon Technology[2], empresa atuante no desenvolvimento de *software*, juntamente com o apoio da FAPESP para pequenas empresas [OLIVEIRA 2004], [FONTANETTE 2004], pesquisaram e definiram uma abordagem para migração gradativa de aplicações legadas. Esta abordagem, que através da identificação e separação do modelo de negócio da tecnologia utilizada na aplicação legada, fornece para as empresas diretivas para o planejamento de diversas estratégias de modernização, respeitando os interesses, necessidades e investimentos de cada empresa.

1.2 Situação Atual

Há várias abordagens e ferramentas existentes na literatura, que apóiam o processo de modernização de aplicações legadas. Contudo, essas abordagens e ferramentas atuais ainda são complexas para o uso amplo na indústria, pois exigem alto conhecimento técnico para serem utilizadas.

O processo manual e ad-hoc empregado pelas empresas é altamente complexo e oneroso, pois muito tempo é gasto na reescrita da aplicação, além do que a aplicação legada deve continuar em manutenção neste período. Desta forma, a modernização, na verdade, consiste na reescrita completa da aplicação em uma nova tecnologia. Este é um processo caro e um investimento de alto risco para a empresa, que se apóia em ferramentas não-integradas e processos ad-hoc, sendo este um dos grandes motivos para que 23% dos projetos de modernização sejam cancelados mesmo antes de concluídos [STANDISH GROUP 2001].

Uma abordagem que vem sendo pesquisada neste sentido é a utilização de um sistema de transformação na reengenharia de aplicações. Dentre estes, os projetos com o ST Draco-PUC vêm sendo desenvolvidos praticamente há mais de 20 anos, tendo obtido resultados reais como na conversão de aplicações nos domínios de linguagem de programa como Progress para Java [NOVAIS 2002], Dataflex para Visual Dataflex [NOGUEIRA 2002], Cobol para C++ [LEITE et al. 1996] e Clipper para Java [JESUS 2000]. Também têm-se experiências em domínios de modelagem, como MDL *(Modeling Domain Language)* do projeto RST [FONTANETTE et al. 2001]. Contudo, as experiências com o ST Draco-PUC têm sido realizadas mais no meio acadêmico.

Atualmente, também existem ferramentas nacionais, como o Apyon Studio [OLIVEIRA 1998], [OLIVEIRA; PALAZZO 1999], [APYON 2000], para especificação de novas aplicações em alto nível

[1] Departamento de Computação - Universidade Federal de São Carlos (UFSCar)- Rod. Washington Luís, Km 235 - Caixa Postal 676 - Cep.13565-905 - São Carlos-SP.
[2] Apyon Technology S/A - Av. Maria Coelho Aguiar, 215 Bloco G – Piso Jardim - Cep. 05805-000 - São Paulo – SP.

de abstração e geração de código em diferentes tecnologias, tais como *Microsoft DNA*, *Microsoft .NET, Java Beans, Java EJB/J2EE,* etc.

As abordagens de modernização e reconstrução de aplicações legadas, o ST Draco-PUC e o Apyon Studio são ferramentas e conceitos que podem ser integrados e oferecidos de uma forma mais prática para apoiar o processo de modernização, obtendo-se uma maior produtividade, com menor esforço e risco para a conversão de aplicações legadas.

Outras abordagens de modernização de aplicações legadas têm sido propostas como [COYLE 2000] e [ALVARO et al. 2003], porém o problema da reconstrução de *software* ainda não foi solucionado de forma satisfatória e normalmente as soluções não oferecem flexibilidade para utilização prática nas empresas.

1.3 Problemas

Além da possibilidade de converter totalmente uma aplicação, existe também a possibilidade de somente adaptá-las para acomodar novas tecnologias, como, por exemplo, oferecer alguns serviços online, motivado pelo amadurecimento das tecnologias para Web.

Outro problema encontrado é que, geralmente, as aplicações legadas não são estruturadas de forma que possam ter partes aproveitadas e acessadas por novos módulos construídos. Desta forma, as empresas optam por reescrever todos os módulos, já utilizando novos conceitos e tecnologias, otimizando seu desempenho e inserindo novos recursos solicitados pelos usuários e que até o momento não puderam ser implementados devido a restrições técnicas ou falta de disponibilidade de recursos.

Há também a dificuldade de algumas abordagens tratarem a necessidade de disponibilizar as partes já convertidas da aplicação e sua comunicação com a aplicação legada, não necessitando aguardar até o final do processo de migração. Esta dificuldade pode ser observada nas abordagens de migração não-gradativas, onde a aplicação é convertida na sua totalidade de uma única vez, como ocorre, por exemplo, no método RST [FONTANETTE et al. 2002a], [FONTANETTE et al. 2002b], [FONTANETTE et al. 2002c], [FONTANETTE et al. 2002d], [FONTANETTE et al. 2002e].

No método RST, o processo de migração de aplicações é realizado através de transformadores que automatizam grande parte da migração. Esses transformadores possuem transformações que mapeiam a sintaxe e semântica dos comandos de um programa, de um domínio, em outro programa, do mesmo ou outro domínio. A construção dessas transformações se torna árdua, demandando conhecimento da sintaxe e semântica dos domínios envolvidos, exigindo muito

tempo do Engenheiro de *Software*, tornando-se uma tarefa muitas vezes inviável no processo de migração somente através de transformações.

Um outro problema é que, uma vez iniciado o processo de migração não-gradativa, o Engenheiro de *Software* só poderá testar, validar e disponibilizar a nova aplicação após sua total migração, o que normalmente é inviável para a empresa.

O Apyon Studio é uma ferramenta que oferece alta produtividade no desenvolvimento de novas aplicações. Desta forma, o ponto de partida do Apyon Studio é o modelo de dados disponível, e a partir destas informações é possível detalhar as demais especificações. Mas este processo é válido para projetos novos e, no caso de aplicações legadas, muitas vezes seus modelos e documentações não existem ou estão completamente desatualizados, devido às manutenções constantes para acréscimo de novas funcionalidades e recursos tecnológicos. Uma aplicação não documentada dificulta a manutenção e o processo de migração, pois o Engenheiro de *Software* precisa entender a aplicação e, neste caso, ele contará apenas com o código legado, não possuindo qualquer outra fonte de informação. Isto pode ocasionar um atraso no processo e elevar o seu custo, além do risco de ambigüidade durante o processo de entendimento do código legado. Desta forma é necessário que a equipe alocada na manutenção da aplicação legada também participe do seu processo de reescrita, mesmo sem conhecer a nova tecnologia e os novos recursos.

1.4 Motivação

Os principais pontos que motivaram a realização deste projeto de pesquisa foram:

- a grande necessidade que as empresas têm de reescrever suas aplicações de forma gradativa, devido à complexidade envolvida e seus orçamentos;
- a possibilidade de dar flexibilidade ao processo de migração, permitindo a migração gradativa de aplicações legadas;
- a possibilidade de transferir para as empresas uma abordagem que permita a modernização de suas aplicações de forma gradativa, reduzindo assim o impacto de uma migração direta;
- a possibilidade de reduzir a tarefa do Engenheiro de *Software* durante a construção dos transformadores no ST Draco-PUC; e
- a oportunidade de oferecer uma solução robusta e prática, que possa ser aplicada em grandes empresas na migração de suas aplicações.

Em um processo de migração gradativa é oferecida a convivência simultânea entre a aplicação legada e as partes já convertidas da aplicação. Essa convivência permite reduzir o impacto de um processo de migração direta, em que a aplicação é convertida e disponibilizada somente no final do processo, causando certo desconforto nos usuários da aplicação.

Neste projeto de pesquisa propôs-se a separação dos aspectos funcionais (regras de negócio) e não-funcionais (referentes à tecnologia) de uma aplicação, oferecendo às empresas a flexibilidade de poder reconstruir as partes referentes à tecnologia, sem converter as regras de negócio. Assim, o Engenheiro de *Software* pode testar e validar a parte reconstruída, uma vez que as regras de negócio, ainda legadas, estão funcionais.

Para tal, as ferramentas ST Draco-PUC e Apyon Studio foram integradas para que o Engenheiro de *Software* tivesse suas tarefas minimizadas, pois a parte referente à geração de tecnologia, que antes era realizada por transformações manuais, agora poderá ser realizada através dos geradores e *templates* disponíveis no Apyon Studio.

Esta integração permite, também, explorar idéias e conceitos, antes tratados somente pelo ST Draco-PUC, como a interpretação de código legado, ou somente pelo Apyon Studio, como o desenvolvimento de novos projetos utilizando geração de código.

Como a conversão de uma aplicação pode exigir a mudança ou inclusão de funcionalidades, conforme as necessidades de seus usuários, este requisito também pôde ser atendido durante o processo de migração. Utilizando o Apyon Studio, esta tarefa é facilitada, devido aos conceitos de especificação abstrata existente na ferramenta. O Apyon Studio, através destas especificações armazenadas em seu repositório, também gera a documentação atualizada da aplicação, facilitando seu entendimento e suas manutenções e modernizações futuras.

Atualmente, muitos projetos de conversão não são iniciados devido à necessidade de disponibilizar equipe e recursos com dedicação exclusiva para esta tarefa, já que, normalmente, não é um processo realizado de forma gradativa e com muita flexibilidade. Contudo, baseado nas idéias de aproveitamento de código existente e na possibilidade de geração semi-automática de código em diversas tecnologias, uma aplicação pode ser convertida por partes ou camadas, de tal forma que as partes convertidas possam se comunicar com as partes não convertidas, até que toda a aplicação seja migrada. Também é possível, por exemplo, reconstruir apenas a parte de interfaces de usuário, convertendo-as para Web, ou ainda disponibilizar rotinas de processamento complexas como *Web Services*.

Por meio da migração gradativa, também é possível gerar aplicações que continuem utilizando bases de dados existentes, permitindo converter módulos da aplicação sem afetar os demais módulos legados que estão em uso na empresa.

A utilização do ST Draco-PUC e do Apyon Studio em uma abordagem gradativa de migração permite às empresas uma maneira viável de migração das aplicações legadas, atendendo às novas tecnologias como JAVA ou .NET, além de facilitar a manutenção e as modernizações futuras. As empresas também conseguirão converter suas aplicações para interfaces com recursos multimídia ou baseadas na *Web*, oferecendo rapidamente novas opções ao mercado.

Este capítulo apresentou uma síntese sobre as abordagens de modernização de aplicações legadas nas empresas, bem como a situação atual deste processo nas empresas e os principais fatores que motivaram este projeto de pesquisa.

1.5 Objetivos

O objetivo principal deste projeto é pesquisar uma abordagem para migração gradativa de aplicações legadas, que possa direcionar as tarefas do Engenheiro de *Software,* reduzindo a complexidade, aumentando a produtividade e minimizando os custos despendidos no processo de migração. Normalmente, as abordagens de migração propõem a migração não-gradativa, ou seja, a migração total e direta da aplicação. Por exemplo, o método RST [ALVARO et al. 2003], [FONTANETTE et al. 2002a], [FONTANETTE et al. 2002b], processo de migração semi-automatizado pelo ST Draco-PUC, faz a conversão de toda a aplicação de uma única vez, através de transformadores que mapeiam os domínios origem para destino.

A abordagem proposta permitirá a migração de aplicações legadas de forma gradativa a fim de tornar esta tarefa mais confiável, rápida e flexível para as empresas, que poderão estudar a melhor forma de migrar suas aplicações, de acordo com suas disponibilidades de tempo e recursos, não precisando aguardar até o final do processo para utilizá-las, podendo conviver com as partes já convertidas da aplicação e as partes ainda legadas.

Considerando que grande parte de uma aplicação está relacionada à tecnologia (interfaces de usuário, páginas *Web*, tratamento de erros, passagem de parâmetros, multi-idiomas, segurança, mapeamento objeto-relacional, controle de sessão, gerenciamento de persistência, validações básicas de dados, controle de transação, gerenciamento de conexão, elaboração de comandos SQL, compatibilidade entre *browsers*, compatibilidade entre sistemas operacionais, entre outros) e uma outra parte, normalmente menor, está relacionada às regras de negócio (cálculos, processos, validações e restrições, entre outros), é possível utilizar os recursos do ST Draco-PUC para identificar e separar tecnologia e negócio, visando simplificar e organizar a manutenção e evolução do aspecto tecnológico das aplicações.

De acordo com estudos existentes [STANDISH GROUP 2001], a parte referente à tecnologia varia entre 70 e 80% de uma aplicação comercial, enquanto que a parte de negócio varia entre 20 e 30%, dependendo do tipo de aplicação. Assim, torna-se importante que a migração gradativa de aplicações legadas possa ser obtida através da migração independente entre tecnologia e negócio.

Essa separação permitirá a migração gradativa da aplicação legada, possibilitando que as partes convertidas da aplicação trabalhem integradas com as partes ainda não convertidas. Pode-se então, por exemplo, migrar apenas a parte referente à tecnologia como, por exemplo, as interfaces de usuário, deixando as regras de negócio inalteradas, acessando os dados no equipamento onde está instalada a aplicação legada, por exemplo, um mainframe. Isso é possível, pois a tecnologia pode ser gerada novamente no Apyon Studio, independente da linguagem de programação utilizada. Em seguida, com a aplicação sendo executada, as regras de negócio são convertidas gradativamente sem perder a integração com as outras aplicações ou módulos que estiverem executando na plataforma legada (mainframe, por exemplo).

Outro objetivo deste projeto é pesquisar a possibilidade de conversão ou reescrita do código relativo ao negócio, aproveitando as principais informações do código legado (código das regras de negócio, módulos da aplicação, base de dados, entre outros), que na maioria das vezes está desestruturado devido às inúmeras correções e ajustes para atender os requisitos do cliente, realizadas ao longo de sua existência.

1.6 Organização da Dissertação

A dissertação está organizada em cinco capítulos, além da seção referente às referências bibliográficas.

O primeiro capítulo contém a introdução com a contextualização do problema, situação atual, motivação e objetivos.

No Capítulo 2 são apresentadas e discutidas algumas das principais abordagens para a modernização de aplicações legadas existentes na literatura e uma síntese com considerações sobre essas abordagens. Baseado nessas considerações são apresentados alguns requisitos necessários para uma abordagem apropriada. São apresentadas também algumas formas de integração e acesso aos dados, alguns possíveis cenários da migração gradativa e as ferramentas de apoio utilizadas neste trabalho.

No Capítulo 3 é apresentada a abordagem AMGraA para modernização de aplicações legadas.

No Capítulo 4 é apresentada a realização de um estudo de caso e seus resultados utilizados para validar a abordagem AMGraA.

No Capítulo 5 são apresentadas as considerações finais abordando uma síntese dos principais resultados, trabalhos correlatos, análise critica e as limitações encontradas no decorrer do trabalho, as publicações e os trabalhos futuros.

No Apêndice é apresentada a documentação do protótipo da ferramenta Migration Project Planning (MPP) construída no decorrer do projeto.

Capítulo 2

Fundamentação Teórica

Neste capitulo são apresentadas e discutidas algumas das principais abordagens para a modernização de aplicações legadas existentes na literatura e uma síntese com considerações sobre essas abordagens. Baseado nessas considerações são apresentados alguns requisitos necessários para uma abordagem apropriada. São apresentadas também algumas formas de integração e acesso aos dados, alguns possíveis cenários da migração gradativa e as ferramentas de apoio utilizadas neste trabalho.

A seção 2.1 apresenta uma visão geral sobre a necessidade de evolução das aplicações legadas. A seção 2.2 apresenta o processo de modernização de *software*, bem como a Reengenharia e apresenta uma visão do legado nas duas últimas décadas. A seção 2.3 apresenta e discute algumas das abordagens de modernização existentes na literatura. A seção 2.4 discute alguns requisitos necessários para uma abordagem de modernização apropriada. A seção 2.5 apresenta algumas formas de integração e acesso aos dados. A seção 2.6 apresenta alguns cenários para a aplicação da migração gradativa. A seção 2.7 apresenta as ferramentas utilizadas neste projeto de pesquisa.

2.1 A Evolução das Aplicações Legadas

Segundo [SEACORD et al. 2003], a partir do momento que uma aplicação de *software* é lançada, começa a corrida contra o tempo. O clichê "aplicação legada é uma aplicação que foi escrita ontem" é totalmente verdadeiro. Hoje, o aumento do ritmo de desenvolvimento de tecnologias dita o ritmo em que esta tecnologia se tornará obsoleta.

Uma aplicação ao longo dos anos, passa por uma série de modificações necessárias para que esta possa atender os requisitos dos usuários e as exigências do mercado. Essas modificações são necessárias para suportar a evolução da aplicação.

A evolução de uma aplicação corresponde a atividades que vão desde a inserção de um novo campo na base de dados até a completa re-implementação da aplicação. As atividades para evolução podem ser divididas em 3 categorias: manutenção, modernização, e substituição ou re-desenvolvimento [WEIDERMAN et al. 1997 apud [SEACORD et al. 2003]][3] .

A Manutenção de *Software* é um processo incremental e repetitivo, onde alterações são feitas na aplicação. Estas alterações envolvem eliminação de erros e melhorias funcionais. A manutenção é necessária para suportar a evolução de qualquer aplicação, mas tem limitações, tais como: adoção de novas tecnologias, pois melhorias, como a implementação de uma arquitetura distribuída, ou uma interface gráfica para *Web*, por exemplo, não são consideradas atividades de manutenção.

A Modernização do *Software* se esforça para evoluir uma aplicação legada, ou elementos da aplicação, quando algumas práticas convencionais como a manutenção e aprimoramento não conseguem mais alcançar algumas propriedades desejadas. A modernização envolve mudanças mais extensas que a manutenção, mas conserva uma porção significante da aplicação existente. As mudanças envolvem a reestruturação das aplicações e inserção de novas funcionalidades.

A Substituição de *Software* ou Re-Desenvolvimento requer a reconstrução da aplicação desde o início, e é apropriada quando a aplicação legada não consegue atender às necessidades do negócio e quando a modernização não é possível ou não vale a pena em relação aos custos [BISBAL et al. 1997 apud [SEACORD et al. 2003]][4] . Na verdade, o risco envolvido é, normalmente, muito grande para as organizações contemplarem uma abordagem de re-desenvolvimento. Outro fator preocupante com este tipo de abordagem é o fato de que a tecnologia e as exigências de negócio estão constantemente mudando. Assim, ao término de um longo processo, a organização poderia terminar com um sistema desenvolvido baseado em uma tecnologia obsoleta, que já não satisfaz suas necessidades empresariais [BISBAL; LAWLESS 2003].

Na Figura 1 é mostrada como várias atividades de evolução são aplicadas em diferentes etapas do ciclo de vida de uma aplicação.

A linha tracejada representa o crescimento das necessidades do negócio. As linhas contínuas representam as funcionalidades atendidas pelas aplicações. A manutenção contínua da aplicação permite atender os requisitos por um determinado tempo, mas conforme a aplicação vai envelhecendo a manutenção não vai ter resultados. Conseqüentemente, a modernização que

[3] WEIDERMAN, H., BERGEY, J., SMITH, D., TILLEY, S.. Approaches to Legacy System Evolution. Pittsburgh, PA: Reengineering Center, Software Engineering Institute, Carnegie Mellon University. (CMU/SEI-97-TR-014). http://www.sei.cmu.edu/activities/cbs/mls/links.html#weiderman97.

[4] BISBAL, J., LAWLESS, D., WU, B., GRINSON, J., WADE, V., RICHARDSON, R., O'SULLIVAN, D.. An Overview of Legacy Information Integration. Proceedings of the 4th Asian-Pacific Software Engineering and International Computer Science Conference (APSEC 97, ICSC 97) Hong Kong, p. 529-530, Dezembro 1997.

demanda mais tempo e esforço que a atividade de manutenção vai ser necessária. Finalmente, quando a aplicação legada não puder ser mais evoluída, ela deverá ser substituída.

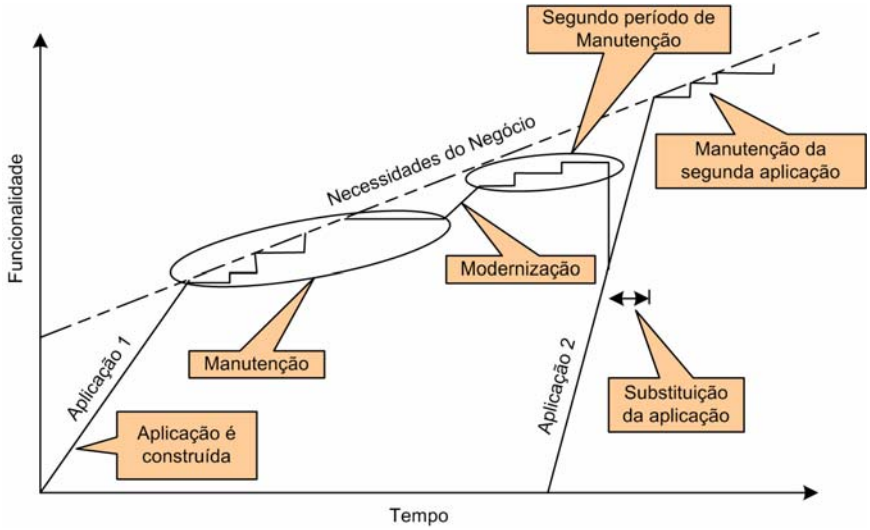


Figura 1 - Ciclo de Vida de uma aplicação [SEACORD et al. 2003]

Determinar as atividades a serem aplicadas de forma apropriada nas diferentes etapas do ciclo de vida da aplicação é um desafio [SEACORD et al. 2003].

As alterações realizadas nas aplicações seguem as seguintes categorias [SEACORD et al. 2003]:

1. **Perfectiva ou de Melhoria**. Estas alterações são feitas para melhorar a aplicação, tais como adicionar uma nova funcionalidade, aumentar o desempenho, usabilidade ou outros atributos da aplicação. Estes tipos de mudanças são chamados de melhorias.
2. **Corretivas**. Estas alterações são feitas para corrigir erros na aplicação.
3. **Adaptativas**. Estas alterações são feitas para adaptar a aplicação a novos ambientes, tais como novos sistemas operacionais, ferramentas, sistemas de gerenciamento de banco de dados e componentes de prateleira.
4. **Preventivas ou Reengenharia**. Estas alterações são feitas para melhorar a manutenibilidade e segurança da aplicação. Ao contrário das três categorias anteriores, esta procura simplificar evoluções futuras da aplicação.

Segundo [MARTIN; MCCLURE 1983], mais de 75% dos custos com manutenção são com atividades do tipo adaptativa e perfectiva, de acordo com dados coletados há mais de 20 anos.

Estudos recentes indicam que esta distribuição ainda não se alterou [NOSEK; PALVIA 1990 apud [SEACORD et al. 2003]][5] e [VLIET; KARK 2000 apud [SEACORD et al. 2003]][6] .

Mudanças no código no decorrer dos anos levam a um código menos manutenível. Conforme a aplicação é continuamente modificada, sua complexidade, que reflete uma estrutura deteriorada, aumenta ao menos que alguma precaução seja tomada para mantê-la ou reduzi-la [LEHMAN; BELADY 1985 apud [SEACORD et al. 2003]][7]. O aumento da complexidade significa que a aplicação se torna mais frágil em relação às mudanças.

O custo total destes tipos de modificações tende a exceder os custos iniciais de desenvolvimento. Os custos relativos à manutenção e evolução da aplicação têm aumentado e representam mais de 90% do custo total [MOAD 1990 apud [SEACORD et al. 2003]][8] e [ERLIKH 2000 apud [SEACORD et al. 2003]][9] .

Empresas com aplicações legadas, com orçamentos limitados, devem obter ótimos retornos em seus investimentos. Para tomar a decisão correta, as empresas devem avaliar suas aplicações legadas, determinar uma estratégia de migração apropriada, e analisar as implicações de cada ação a ser tomada.

Assim, considerando as três atividades apresentadas para a evolução de uma aplicação de *software*, neste projeto de pesquisa o foco foi dado para a atividade de modernização de *software*, uma vez que este trabalho propõe uma abordagem para a modernização de aplicações legadas.

2.2 Modernização de *Software*

A Modernização é usada quando a aplicação legada requer mudanças mais extensas e significativas do que aquelas realizadas na manutenção, mas precisa preservar o negócio da aplicação. Os motivos para a modernização de uma aplicação geralmente derivam da fragilidade e pouca consistência da aplicação legada, da falta de flexibilidade, isolamento e pouca capacidade de extensão.

A modernização pode ser distinguida pelo nível de compreensão da aplicação exigida para o esforço da modernização [WEIDERMAN 1997]. Há dois tipos:

- **Caixa Branca**: que exige um conhecimento sobre o código interno da aplicação legada. Se não há este conhecimento, então é necessário passar pelo processo de compreensão

[5] NOSEK, J., PALVIA, P.. Software Maintenance Management; Changes in the Last Decade. Journal of Software Maintenance: Research and Practice 23: p. 157-174, 1990.
[6] VLIET, H., KARK, A.. Software Engineering: Principles and Practice. West Sussex, England: Willey, 2000.
[7] LEHMAN, M. M, L. BELADY. Program Evolution: Process of Software Change. London. Academic Press, 1985.
[8] MOAD, J.. Maintaining the Competitive Edge. Datamation 61-62, 64-66. 2000.
[9] ERLIKH, L. Leveraging Legacy Systems Dollars for E-Business. IEEE. IT Pro May/June, p. 17-23, 2000.

do código da aplicação. Este processo envolve a modelagem do domínio da aplicação, a extração de informações do código e a criação de abstrações que descrevem a estrutura da aplicação. Após a análise e compreensão do código, este tipo de modernização, normalmente, inclui a reestruturação da aplicação ou do código. A reestruturação da aplicação é a transformação de uma forma de representação para uma outra no mesmo nível de abstração, preservando o comportamento do sistema (funcionalidade e semântica). É usada para aumentar a manutenibilidade e o desempenho da aplicação, e;

- **Caixa Preta**: que exige conhecimento apenas das interfaces externas da aplicação legada. Envolve examinar as entradas e saídas de uma aplicação legada, para entender as interfaces da aplicação. Este tipo não é tão trabalhoso e difícil quanto o tipo caixa branca. Este tipo de modernização é, geralmente, baseado na técnica de *wrapping* – onde a aplicação legada é envolvida por uma camada de *software* que esconde a complexidade da aplicação antiga e exporta uma interface mais moderna. *Wrapping* é uma tarefa caixa preta, uma vez que somente a interface legada é analisada e o código interno é ignorado. Esta solução nem sempre é prática e com freqüência exige a compreensão dos módulos internos, fazendo uso de técnicas caixa branca.

Uma pesquisa do Standish Group mostra que muitas das tentativas de modernização de *software* falham e que 23% dos projetos são cancelados antes de terminarem, enquanto que 28% terminam no tempo e orçamentos previstos e com a funcionalidade esperada [STANDISH GROUP 2001].

Um dos desafios para a modernização é o tamanho da aplicação. Segundo [ULRICH 1990], a quantidade de código legado é imensa e crescente. Isto ficou evidente no final dos anos 90, quando as organizações contrataram um grande número de programadores Cobol e consultores para corrigir os problemas relacionados ao ano 2000. Em 1990, 120 bilhões de linhas de código – Cobol e Fortran - estavam sofrendo manutenções.

Segundo [SOMMERVILLE 2001], uma estimativa é que aproximadamente 250 bilhões de linhas de código estão agora em manutenção e este número está aumentando o tempo todo.

As aplicações de *Software* tendem a expandir com o tempo, mas raramente algum capital é empregado para remover códigos inutilizáveis [SEACORD et al. 2003]. Mais *software* significa mais *software* para evoluir e manter.

Uma forma de modernização para melhorar a capacidade e manutenibilidade de uma aplicação legada, introduzindo tecnologias e práticas modernas, é a Reengenharia de *Software*, que será apresentada a seguir.

2.2.1 Reengenharia de *Software*

A Reengenharia de *Software* oferece uma abordagem disciplinada para migrar aplicações legadas para aplicações evolutivas, ou seja, aplicações desenvolvidas em uma arquitetura flexível que permita evoluções. Este processo aplica os princípios da Engenharia de *Software* em uma aplicação existente para atender a novos requisitos [SEACORD et al. 2003].

Em meados dos anos 90, o SEI *(Software Engineering Institute)* definiu a Reengenharia como uma transformação sistemática de uma aplicação existente para uma nova forma, realizando melhorias na operação, na funcionalidade, no desempenho, ou na capacidade de evoluir para o novo sistema com menores custos, prazos e riscos [TILLEY; SMITH 1995 apud [SEACORD et al. 2003]][10] .

Chikofsky em [CHIKOFSKY; CROSS 1990] apresenta uma terminologia empregada na Engenharia de *Software* para referenciar as tecnologias de análise e entendimento de aplicações legadas, com o objetivo de racionalizar termos que já estão em uso. Os principais termos definidos e relacionados são: Reengenharia, Engenharia Reversa, Engenharia Avante, Redocumentação, Recuperação de Projeto e Reestruturação. O relacionamento entre esses termos é mostrado na Figura 2, considerando-se que o ciclo de vida da aplicação possui três grandes etapas: Requisitos, Projeto e Implementação, com claras diferenças no nível de abstração. Os Requisitos tratam da especificação do problema, incluindo objetivos, restrições e regras de negócio. O Projeto trata da especificação da solução. A Implementação trata da codificação, teste e entrega da aplicação em operação.

Na Figura 2 é mostrada a direção seguida pela Engenharia Avante, do nível de abstração mais alto para o mais baixo. A Engenharia Avante é o tradicional processo de desenvolvimento, em que se parte de alto nível de abstração e

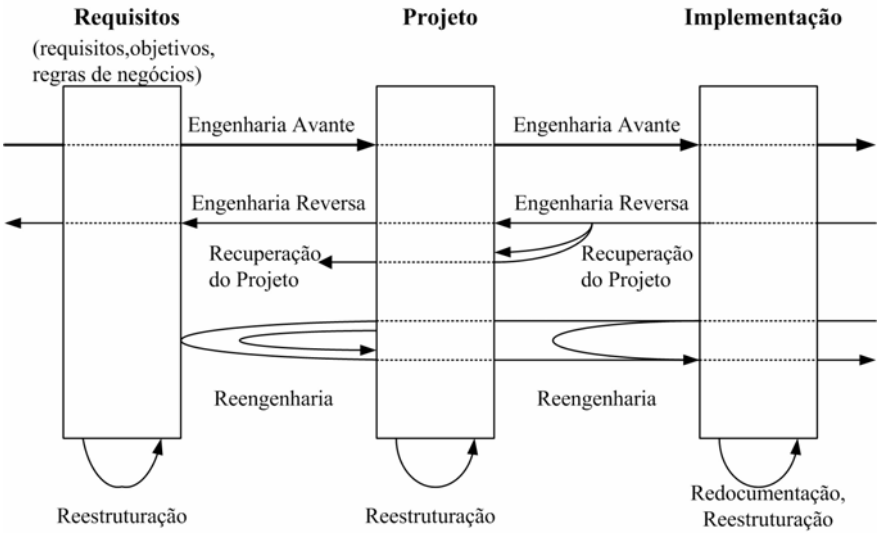


Figura 2 - Relação entre os termos [CHIKOFSKY; CROSS 1990]

[10] TILLEY, S. R., SMITH, D. Perspectives on Legacy System Reengineering. Pittsburgh, PA: Reengineering Center, Software Engineering Institute, Carnegie Mellon University. http://www.sei.cmu.edu/activities/cbs/mls/links.html#tilley95.

lógica, passando pela análise de requisitos e projeto até a implementação física da aplicação [CHIKOFSKY; CROSS 1990]. A Figura mostra também, que a Engenharia Reversa percorre o caminho inverso, podendo utilizar-se da Recuperação de Projeto para melhorar o nível de abstração. A Reengenharia geralmente inclui uma Engenharia Reversa, seguida de alguma forma de Engenharia Avante ou Reestruturação.

O objetivo da Reengenharia de *Software* é manter o conhecimento adquirido com as aplicações legadas e utilizar estes conhecimentos como base para a sua evolução contínua, implementando inovações tecnológicas, novos requisitos e corrigindo erros. O código legado possui técnicas de programação, decisões de projeto, requisitos do usuário e regras de negócio que podem ser recuperados e reconstruídos sem perda da semântica.

Segundo [CHIKOFSKY; CROSS 1990], a Reengenharia tem por finalidade examinar e alterar uma aplicação existente para reconstituí-la em uma nova forma e depois implementá-la na nova forma. A Reengenharia tem como objetivo principal melhorar a qualidade global da aplicação, mantendo, em geral, as funções da aplicação existente. Mas, ao mesmo tempo, podem-se adicionar novas funções e melhorar o desempenho [PRESSMAN 1995]. Segundo Jacobson [JACOBSON; LINDSTROM 1991], "A Reengenharia é o processo de criar uma descrição abstrata da aplicação, elaborar mudanças em alto nível de abstração e então implementá-las na aplicação".

Ainda segundo [CHIKOFSKY; CROSS 1990], no *software*, a Engenharia Reversa é o processo de análise de uma aplicação para:

- Identificar seus componentes e inter-relacionamentos e;
- Criar representações da mesma em outra forma ou em um nível mais alto de abstração.

O objetivo de sua utilização é criar uma descrição mais abstrata da aplicação existente, partindo do código fonte, para recuperar ou recriar seu projeto e decifrar os requisitos que foram implementados, visando auxiliar na manutenção e adaptar a aplicação às novas plataformas.

As informações extraídas do código fonte, através da Engenharia Reversa, podem estar em diversos níveis de abstração. O ideal seria ter um nível de abstração mais alto possível [PRESSMAN 1995].

Há várias subáreas da Engenharia Reversa, sendo a Redocumentação e a Recuperação do Projeto, as mais utilizadas.

A Re-Documentação é a criação de uma representação do programa que seja equivalente e esteja no mesmo nível de abstração. Muitos a consideram uma forma de Reestruturação. O objetivo é recuperar uma documentação que existiu ou deveria ter existido da aplicação. Algumas

ferramentas utilizadas aqui são os *prettyprinters*, que mostram a listagem do código numa forma aprimorada; os geradores de diagramas, que criam diagramas a partir do código, mostrando o fluxo de controle e geradores de listagem de referência cruzada.

A Recuperação do Projeto usa conhecimentos sobre o domínio de aplicação, informações externas, dedução ou raciocínio para a compreensão da aplicação em questão, de forma a identificar maiores e mais significativos níveis de abstração entre aqueles obtidos no exame da própria aplicação [CHIKOFSKY; CROSS 1990].

Uma das maiores aplicações da Engenharia Reversa é a Manutenção de *Software*. A Manutenção de *Software* é a modificação de uma aplicação, depois da entrega, para corrigir falhas, melhorar o desempenho e outros atributos, ou para adaptar o produto a mudanças de ambiente [CHIKOFSKY; CROSS 1990]. É uma tarefa difícil e custosa, pois, geralmente, os mantenedores do sistema não participaram do seu desenvolvimento. Portanto, será necessário a estas pessoas examinar e aprender sobre a aplicação. Neste contexto, a Engenharia Reversa é a parte da manutenção que vai ajudar no entendimento da aplicação para que as mudanças possam ser realizadas., melhorando a compreensibilidade da aplicação, através da documentação produzida.

A Engenharia Reversa deve produzir documentos que ajudem a aumentar o conhecimento geral de aplicações de *software*, facilitando o reuso, manutenção, teste e controle de qualidade da aplicação e isto deve ser feito de forma automática, de preferência, devido à quantidade de linhas de código envolvidas.

2.2.2 Visão Geral da Aplicação Legada: Duas Décadas

Segundo [COYLE 2000], em meados dos anos 80, as estratégias para lidar com os legados focavam no entendimento dos programas e na extração das funcionalidades essenciais do código existente. O objetivo da migração era desenvolver novos programas, que fossem mais robustos e manuteníveis que seus antecessores. Os programadores se esforçaram para converter as aplicações legadas para arquiteturas cliente-servidor de 2-camadas com dados que residiam em um servidor enquanto a lógica do negócio era armazenada no cliente. Porém, logo ficou aparente que arquiteturas simples de 2-camadas não suportariam o crescente tráfego de rede, sem falar na problemática dos clientes "gordos", ou seja, clientes sobrecarregados.

No início dos anos 90, a arquitetura de 3-camadas substituiu a de 2-camadas, fornecendo mais flexibilidade e colocando menos responsabilidades no cliente. A primeira camada, a interface do usuário, era o *front-end* da aplicação. A lógica de negócio ficava na segunda camada, onde existia a maior parte da programação. Esta camada, normalmente, possuía um

monitor de transação para administrar interações complexas com a terceira camada de dados, normalmente um banco de dados relacional. Ainda, embora a arquitetura 3-camadas tenha desfrutado de um considerável sucesso controlando o aumento do tráfego na rede, o custo de analisar e reestruturar código legado permaneceu alto [COYLE 2000].

Em meados dos anos 90, as linguagens orientadas a objetos e *frameworks* de componentes amadureceram, mudando o foco da migração do legado para integração do legado. O amadurecimento e a aceitação das linguagens orientadas a objetos abriram as portas para os *"wrappers"* capazes de esconder partes internas do legado, enquanto fornece uma visão de interface bem definida para o mundo externo.

Dadas as desvantagens de se desenvolver uma nova aplicação, muitas empresas são forçadas a buscar modos alternativos para lidar com suas aplicações legadas. As soluções mais práticas focam na técnica de *"wrapping"* o qual envolve os dados existentes, programas individuais, sistemas de aplicação e interfaces com novas interfaces. Em essência, isto dá aos componentes antigos, novas operações ou uma "nova e aperfeiçoada" aparência [WEIDERMAN et al. 1997 apud [BISBAL; LAWLESS 2003][11]]. Os componentes envolvidos agem como um servidor, executando alguma função requerida por um cliente externo que não precisa saber como o serviço é implementado [SNEED 1996 apud [BISBAL; LAWLESS 2003][12]]. Esta técnica permite que as organizações reusem componentes bem-testados que eles confiam.

A implementação mais usada de *wrapping* é o *screen scraping* que substitui o *front-end* de um sistema legado baseado em caráter por uma interface gráfica [BENNETT 1995 apud [BISBAL; LAWLESS 2003]][13]. Esta implementação barata de uma GUI permite aos usuários empregarem ferramentas de manipulação de dados para introduzir os dados e processar a saída do sistema.

Apesar do sucesso comercial do *screen scraping*, ela é ainda uma solução de curto prazo. Implementar uma GUI em um sistema legado não trata muitos dos sérios problemas que tais sistemas enfrentam, como sobrecarga, funcionalidade estática e altos custos de manutenção. Em muitos casos, o *screen scraping* de fato se mistura aos problemas de manutenção da organização, uma vez que ele também vai precisar sofrer manutenções.

Passado os anos 90, o ano 2000 nos forçou a reestruturar, quebrar e melhorar muitos programas legados mal-escritos e mal-documentados. A Internet tem obrigado as companhias a anteceder a reescrita cara e demorada dos softwares. As companhias estão adicionando *browsers*

[11] WEIDERMAN, N., et al. Implications of Distributes Object Technology for Reengineering, Tech. Report CMU/SEI-97-TR-005, Carnegie Mellon Univ., Pittsburgh, 1997.
[12] SNEED, H. M., Encapsulating Legacy Software for Use in Client/Server Systems, Proc. Third Working Conf. Reverse Eng., IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp.104–119.
[13] BENNETT, K., "Legacy Systems," IEEE Software, Jan. 1995, pp.19–73.

como *front-ends* em sistemas existentes ou criando novas aplicações que taticamente aumentam a funcionalidade do sistema.

A larga disponibilidade dos dados fornecida pelos servidores *Web* abriu novas possibilidades para a tomada de decisão, e as tecnologias de *Data Warehousing* e mineração de dados contribuíram para este grande e novo interesse nos dados. Com isso, a *Extensible Markup Language*, XML[14], uma linguagem de marcação extensível, ganhou uma rápida aceitação como um formato de representação de dados universal com larga aplicabilidade para troca de dados distribuída.

2.3 Abordagens de Modernização Existentes

A importância das aplicações legadas e a necessidade que elas têm de oferecer altos níveis de qualidade são reconhecidos pela ampla literatura existente nestes tópicos. Um grande número de abordagens e métodos foi proposto ao longo dos anos na tentativa de enfrentar o problema e auxiliar o processo de modernização dessas aplicações: Os trabalhos propostos por Jacobson e Lindstron em [JACOBSON; LINDSTRON 1991], Markosian em [MARKOSIAN et al. 1994], Wilkening em [WILKENING et al. 1995], Sneed em [SNEED 1996], Olsem em [OLSEM 1998] são somente alguns exemplos.

Em [JACOBSON; LINDSTRON 1991], os autores apresentam uma técnica para efetuar a reengenharia de aplicações legadas, implementadas em uma linguagem procedural como C ou Cobol, obtendo aplicações orientadas a objetos. Os autores mostram como realizar a reengenharia de forma gradual, pois consideram impraticável substituir uma aplicação antiga por uma completamente nova (o que exigiria muitos recursos). Consideram três cenários diferentes: no primeiro se faz mudança de implementação sem mudança de funcionalidade; no segundo se faz a mudança parcial da implementação sem mudança de funcionalidade; e no terceiro se faz alguma mudança na funcionalidade. Os autores utilizaram uma ferramenta CASE específica para orientação a objetos, o "ObjectOry". Os autores optaram pela orientação a objetos como maneira de modernizar o sistema.

Em [MARKOSIAN et al. 1994], Markosian e os demais autores reclamam da falta de apoio computadorizado para a reengenharia de aplicações, em contraposição à grande proliferação de ferramentas CASE *(Computer Aided Software Engineering)* para desenvolvimento de *software*. Os autores afirmam que as ferramentas de transformação da época eram muito limitadas, sendo difíceis suas adaptações a um projeto em particular. Eles abordam uma nova tecnologia para

[14] Extensible Markup Language (XML) - http://www.w3.org/XML/, Acessado em 15/05/2004.

reengenharia, chamada de "tecnologia facilitadora", relatando resultados práticos bastante animadores quanto à produtividade da sua aplicação. A "tecnologia facilitadora" consiste no rápido desenvolvimento de ferramentas para analisar e modificar aplicações legadas. Deve ser usada em tarefas complexas de reengenharia, que estejam sendo feitas de forma manual ou semi-automatizada.

Em [GALL, KLÖSH 1994], os autores afirmam que a transformação de aplicações procedurais em aplicações orientadas a objetos é um processo importante para aumentar o potencial de reuso das aplicações procedurais, mas que existem problemas difíceis de resolver no paradigma procedural, como é o caso da interconexão de módulos. Relatam que o reuso está assumindo papel relevante na produção industrial de *software*, pois reduz os custos de desenvolvimento de *software* e melhora a qualidade do produto final. Os autores propõem um processo de transformação, denominado método COREM *(Capsule Oriented Reverse Engineering Method)* em [GALL, KLÖSH 1993], que usa conhecimento do domínio da aplicação. Esse método consiste de quatro passos principais: recuperação do projeto, modelagem da aplicação, mapeamento dos objetos e transformação do sistema. Embora a aplicação de ferramentas seja útil e necessária durante o processo de transformação COREM, é imprescindível a aquisição de conhecimento adicional pelo Engenheiro de *Software*. Portanto, a automação completa desse processo de transformação não é possível. Mas muitos passos do processo podem ser auxiliados por ferramentas.

Em [WILKENING et al. 1995], Wilkening e os demais autores apresentam um processo para efetuar a reengenharia de aplicações legadas, aproveitando partes de sua implementação e projeto. Esse processo inicia-se com a reestruturação preliminar do código fonte, para introduzir algumas melhorias, como remoção de construções não-estruturadas, código "morto" e tipos implícitos. A finalidade dessa reestruturação preliminar é produzir um programa fonte mais fácil de analisar, entender e reestruturar.

Em seguida, o código fonte produzido é analisado e são construídas suas representações em níveis mais altos de abstração. Com base nessas representações, pode-se prosseguir com os passos de reestruturação, reprojeto e redocumentação, que são repetidos quantas vezes forem necessárias para se obter a aplicação totalmente reestruturada. Pode-se, então, implementar o programa na linguagem destino e dar seqüência aos testes que verificarão se a funcionalidade não foi afetada.

Wilkening apresenta a ferramenta RET, que automatiza parcialmente esse processo, sendo, portanto, uma reengenharia assistida por computador.

Em [PENTEADO 1996], a autora apresenta o método de engenharia reversa Fusion/RE, que se destaca em trabalhos de migração de aplicações. O método propõe algumas diretrizes para obter o entendimento e revitalizar a estrutura do código da aplicação legada, segundo o paradigma orientado a objetos, visando reutilizar a funcionalidade do código legado na reconstrução da aplicação. Essas diretrizes são aplicadas manualmente, com o auxílio da ferramenta *Legacy Aid*.

Em [SNEED 1996], o autor descreve um processo de reengenharia apoiado por uma ferramenta para extrair objetos a partir de programas existentes em COBOL. Ressalta a predominância da tecnologia de objetos na época, principalmente em aplicações distribuídas com interfaces gráficas, questionando a necessidade de migração de aplicações legadas para essa nova tecnologia. Identifica obstáculos à reengenharia orientada a objetos como, por exemplo, a identificação dos objetos, a natureza procedural da maioria das aplicações legadas, que leva a blocos de código processando muitos objetos de dados, a existência de código redundante e a utilização arbitrária de nomes. Assume como pré-requisitos para a reengenharia orientada a objetos a estruturação e modularização dos programas, além da existência de uma árvore de chamadas da aplicação. É apresentado um processo de reengenharia orientada a objetos, composto de cinco passos: seleção de objetos, extração de operações, herança de características, eliminação de redundâncias e conversão de sintaxe. A seleção de objetos é feita com apoio da ferramenta, mas o responsável pela determinação dos objetos é o Engenheiro de *Software*. A extração de operações particiona a aplicação em sub-rotinas, removendo os segmentos referentes a um determinado objeto e substituindo-os por envio de mensagens ao objeto no qual os dados locais estão encapsulados. Foram usadas as técnicas de particionamento da aplicação em sub-rotinas para obtenção dos métodos e as de substituição destas sub-rotinas por mensagens de acionamento das operações.

Em [OLSEM 1998], o autor propõe a reengenharia gradativa como solução para os problemas da reengenharia de aplicações. O problema da reengenharia de aplicações, segundo Olsem, está em como ela é realizada, isto é, alterando toda a aplicação (plataformas, dados, aplicação e interface), de uma única vez e forma. Para Olsem, aplicações legadas são formadas por quatro classes de componentes (*Software*/Aplicações, Arquivo de Dados, Plataformas e Interfaces) que não podem ser tratados da mesma forma. A reengenharia gradual proposta pelo autor efetua o processo em componentes separados, utilizando estratégias diferentes para cada classe de componentes, diminuindo as probabilidades de fracasso no processo. Duas formas de reengenharia são propostas: com reintegração, no qual os módulos submetidos à reengenharia são reintegrados a aplicação, e sem reintegração, no qual os módulos submetidos à reengenharia são identificados, isolados e farão interface com os programas que não foram submetidos ao

processo por meio de um mecanismo chamado *"Gateway"*, com isso, a aplicação resultante terá componentes que foram submetidos à reengenharia e outros que não o foram.

Em [RIVA 2000], o autor propõe uma abordagem de engenharia reversa de arquitetura de *software*. O autor afirma que o objetivo é aumentar a compreensão da aplicação a fim de auxiliar a manutenção ou um novo desenvolvimento. O autor define "arquitetura" como sendo uma coleção de decisões de projeto que habilitam a implementação de requisitos funcionais e satisfazem aos parâmetros de qualidade da aplicação. É uma clara especificação dos elementos que são envolvidos, como eles interagem e como são usados para satisfazer os requisitos desejados.

Em [FONTANETTE et al. 2002d], apresentamos o método RST, que propõe uma abordagem para a conversão de aplicações através de transformações de *software*. O processo é semi-automatizado por transformadores. Ao recuperar o projeto orientado a objetos, o Engenheiro de *Software* utiliza uma ferramenta CASE para reespecificar a aplicação para realizar refinamentos e re-projetar a aplicação em uma nova arquitetura, por exemplo, orientada a componentes. A implementação do código pode ser realizada através da ferramenta CASE ou através dos transformadores de geração de código. O processo proposto pelo método RST é um processo de migração direta. Este processo não trata a conversão por partes de uma aplicação.

Em [ALVARO et al. 2003], os autores apresentam um ambiente para a reengenharia de *software* baseada em componentes, chamado Orion-RE. O ambiente utiliza técnicas de reengenharia de *software* e de desenvolvimento baseado em componentes para reconstruir aplicações legadas, reutilizando a documentação disponível e o conhecimento embutido em seu código fonte. O Orion-RE integra diferentes ferramentas na execução do processo de reengenharia: um sistema transformacional de *software*, uma ferramenta de modelagem e desenvolvimento, um repositório de artefatos de *software* e uma plataforma de *middleware*. O modelo de processo de *software* guia a utilização do ambiente por meio de uma engenharia reversa para recuperar o projeto do sistema legado, e de uma engenharia avante no qual o sistema é reconstruído usando modernas técnicas de engenharia de *software* como, por exemplo, padrões, *frameworks*, técnicas de Desenvolvimento Baseado em Componentes e *middleware*.

Em [ZOU; KONTOGIANNIS 2003], os autores apresentam um *framework* genérico para apoiar a reengenharia de aplicações legadas procedurais para plataformas orientadas a objetos, através de transformação de código para apoiar a migração gradual. Primeiro, um *framework* de representação de código que usa um modelo de domínio genérico para linguagens procedurais permite a representação de Árvores de Sintaxe Abstratas como documentos XML. Segundo, um conjunto de transformações permite a identificação de modelos de objeto em partes específicas

do código legado. Deste modo, o processo de migração é aplicado gradualmente em diferentes partes da aplicação. Uma técnica de *clustering* é usada para decompor uma aplicação em um conjunto de componentes menores que são satisfatórios para o processo de migração gradual. Finalmente, o processo de migração compõe os modelos de objetos obtidos em cada fase para gerar um modelo de objetos único para a aplicação.

Na Tabela 1 é apresentado um comparativo entre as abordagens discutidas. Foram analisados os seguintes itens:

Tabela 1 - Comparativo das abordagens existentes

Abordagem	Recuperação de Modelos	Reestruturação de Código	Ferramentas de Apoio	Gradativa
Jacobson Lindstron	✓	×	✓	✓
Markosian	✓	×	✓	×
Gall e Klösh	✓	×	✓	×
Wilkening	✓	✓	✓	×
Penteado	✓	×	✓	×
Sneed	✓	✓	✓	×
Olsem	✓	✓	✓	✓
Riva	✓	×	✓	×
Fontanette	✓	×	✓	×
Álvaro	✓	×	✓	×
Zou e Kontogianni	✓	×	✓	✓

- **Recuperação de Modelos**: envolve a recuperação de representações em alto nível de abstração. Estas representações são modelos que redocumentam a aplicação legada. Podem ser recuperados diagramas de fluxo de dados, seqüência de chamadas, diagrama de entidade e relacionamento, diagrama de casos de uso, entre outros;
- **Reestruturação do Código**: envolve modificações no código da aplicação legada para estruturar, eliminar código "morto", facilitando assim o entendimento da aplicação e manutenções futuras;
- **Ferramentas de Apoio**: envolve a utilização de alguma ferramenta de apoio durante o processo;
- **Gradativa**: o processo de modernização é realizado por módulos da aplicação.

2.3.1 Considerações sobre as Abordagens Apresentadas

Para preservar a importância da aplicação legada, a familiaridade que os mantenedores e usuários têm com tal aplicação, e a execução das funcionalidades existentes durante o processo

de modernização, a aplicação precisa ser migrada de forma gradativa [BIANCHI et al. 2003], [SEACORD et al. 2003]. Percebe-se também que cada módulo precisa ser migrado dentro de um curto período de tempo, possibilitando converter os módulos de acordo com as prioridades de negócio. A migração gradativa permite que a aplicação legada continue sendo usada enquanto é convertida, produzindo resultados mais rápidos para os usuários e mantendo a integração necessária para as aplicações ficarem sempre sincronizadas.

Muitas das abordagens não oferecem uma migração gradativa. A maior parte delas trata somente a mudança para o paradigma orientado a objetos [JACOBSON; LINDSTRON 1991], [MARKOSIAN et al. 1994], [GALL; KLÖSH 1994], [SNEED 1996], [PENTEADO 1996], [RIVA 2000], obtendo alguma representação em um nível mais alto de abstração.

Em [GALL; KLÖSH 1994], os autores recuperam diagramas de fluxo de dados (DFD's), diagrama de entidade e relacionamento (DER) e um modelo orientado a objetos da aplicação. Já as abordagens mais recentes, como em [FONTANETTE et al. 2002b], [ALVARO et al. 2003] recuperam diagramas de casos de uso e diagramas de seqüência da aplicação.

Gall e Klösh também apresentam uma técnica de encapsulamento proposta pelo método COREM. O encapsulamento pode muitas vezes ser tão complexo e exigir um grande número de mudanças nos programas principais e periféricos, que é quase uma reescrita da aplicação. O valor gasto pode ser igual, já que uma tecnologia nova também acompanha ferramentas novas de desenvolvimento rápido, agilizando o processo.

Em [WILKENING et al. 1995], o processo pode ser tão complexo e exigir muita mão-de-obra especializada para entender e documentar a aplicação, que é quase uma reescrita da aplicação. E muitas vezes não adianta apenas entender o que existe, necessitando também entender o negócio, que também pode ter mudado, pois existe um *backlog* enorme de solicitações de usuários e existem vários erros conhecidos. Outros fatores como mudanças de decisões da empresa ou lançamento de produtos diferentes, sem contar a globalização e mudanças na economia do país, que freqüentemente ocorrem, exigem mudanças nas aplicações.

As abordagens normalmente contam com pelo menos uma ferramenta de apoio para auxiliar no processo. Em [FONTANETTE et al. 2002b], [ALVARO et al. 2003] os autores sugerem a utilização de transformadores de *software* e uma ferramenta CASE. Contudo, a escrita destes transformadores é manual e exige alto grau de conhecimento técnico do mecanismo de transformação e das sintaxes e semânticas das linguagens de origem e de destino. As abordagens propostas pelo método RST e Orion-RE são abordagens de migração direta, pois não tratam a conversão de uma aplicação por módulos. Portanto, não há uma flexibilidade de

planejar a migração, de estudar o que poderia ser migrado primeiro, por exemplo, para atender as prioridades da empresa.

Uma outra característica das abordagens apresentadas é que elas assumem como aplicação legada as aplicações procedurais. Mas, hoje com o aumento do ritmo do desenvolvimento de novas tecnologias, uma aplicação desenvolvida recentemente pode ser uma aplicação legada [20]. Portanto, as abordagens devem tratar estas aplicações.

A maioria dessas abordagens, tais como [JACOBSON; LINDSTRON 1991], [MARKOSIAN et al., 1994], [GALL; KLÖSH 1994], [PENTEADO 1996], [FONTANETTE et al. 2002b], [ALVARO et al. 2003] visam a mudança do paradigma, do procedural para o orientado a objetos, também recuperando modelos para facilitar o entendimento da aplicação. A recuperação de modelos é importante, pois facilita o entendimento e as manutenções futuras da aplicação. Entretanto, poucas abordagens realizam uma reestruturação de código, que é extremamente importante para eliminar trechos de código inutilizáveis e facilitar a manutenção da aplicação, com exceção das abordagens de Wilkening, Sneed e Olsem [WILKENING et al 1995], [SNEED 1996], [OLSEM 1998].

Embora existam várias abordagens e ferramentas na literatura, elas ainda são complexas para o uso amplo na indústria, pois exigem alto conhecimento técnico e possuem algumas limitações, como a falta de flexibilidade no processo de migração e de integração com outras ferramentas, fazendo com que as empresas realizem o processo de forma manual.

As propostas de modernização realizadas de forma manual, muitas vezes desencorajam as empresas ou até mesmo inviabilizam sua execução devido aos altos custos e a necessidade de realocar pessoas que realizam a manutenção para trabalharem no desenvolvimento da nova aplicação.

A modernização manual também é muitas vezes inviabilizada pelo risco da reescrita de uma aplicação totalmente nova, pois o novo código pode não ter as mesmas funcionalidades (o código legado pode ter rotinas complexas não expostas, de difícil entendimento e análise ou uma série de manutenções sobre manutenções, modificando várias vezes as mesmas informações) ou até mesmo ser finalizado com muitos erros devido à complexidade e falta de experiência da equipe nas novas tecnologias.

Além dos problemas citados acima, essas abordagens não apresentam resultados práticos rápidos para os usuários finais, desmotivando e até mesmo sendo cancelados devido a mudanças de prioridade na empresa [STANDISH GROUP 2001].

2.4 Requisitos de uma Abordagem Apropriada

Através da análise das abordagens existentes, e considerando as variáveis e atributos que uma abordagem deve oferecer para a modernização de aplicações legadas, apresenta-se primeiramente um comparativo entre os dois diferentes tipos de abordagens: gradativa e não-gradativa e em seguida a identificação de alguns requisitos para uma abordagem ideal para a modernização de aplicações legadas.

Considere as seguintes variáveis e atributos:

- **Flexibilidade**: envolve a flexibilidade oferecida pelo processo de modernização. Mudanças podem acontecer no decorrer do projeto, com isso, deve ser analisado o comportamento do processo em relação a estas mudanças, ou seja, se elas serão aceitas ou não;
- **Resultados em curto prazo**: analisa o tempo de espera para apresentação dos resultados do processo de modernização;
- **Mitigação dos riscos**: analisa o poder de amenização dos riscos durante o processo. Por exemplo, se um erro ocorrer, qual será o grau de dificuldade de recuperação;
- **Equipes pequenas**: analisa o número de pessoas necessárias no decorrer do processo de modernização para que os resultados sejam apresentados em um prazo satisfatório;
- **Convivência entre módulos e aplicações**: analisa a convivência, ou seja, a comunicação entre os módulos já migrados e a aplicação legada. É analisada também a capacidade de integração destes módulos com demais aplicações;
- **Planejamento e Controle**: analisa o grau de dificuldade envolvida no planejamento e controle do processo de modernização.

Com base nestas variáveis, na Tabela 2 é mostrado um comparativo entre as abordagens gradativas e não-gradativas.

Tabela 2 – Comparativo Abordagem Gradativa versus Não-Gradativa

	Gradativa	Não-Gradativa
Flexibilidade	☺	😐
Resultados em curto prazo	☺	😐
Mitigação dos riscos	☺	☹
Equipes pequenas	☺	😐
Convivência entre módulos e aplicações	☺	☹
Planejamento e Controle	😐	😐

☺ Atende 😐 Atende Parcialmente / Com Dificuldades ☹ Não-Atende

Tem-se como exemplo a análise da variável flexibilidade. A direção das decisões dentro de uma empresa é sempre dinâmica. Assim, pressões externas e internas podem alterar as metas e a direção do projeto de migração de ponta a ponta. Portanto, a abordagem de migração deve lidar com estas implicações. A abordagem deve ser flexível e permitir à empresa melhor redirecionar os esforços de um processo de migração em resposta a estas mudanças. Isto é um fator positivo nas abordagens gradativas, enquanto que as não-gradativas não apresentam esta flexibilidade.

Conforme as tecnologias mudam, a abordagem gradativa pode fazer adaptações mais facilmente, minimizando o risco de perda dos recursos investidos em processos de migração anteriores.

Os resultados em uma abordagem gradativa são mais imediatos e concretos. Há uma nítida melhoria no tempo necessário para a manutenção, e também para o retorno da aplicação quando for para manutenção. Considerando a perspectiva do usuário, não é necessário aguardar muito tempo para ver algum resultado. À medida que os módulos são migrados, estes são disponibilizados, assim, não há aquela impaciência na espera da nova aplicação. Os resultados em curto prazo e retorno mais imediato em investimento (ROI) agradam a empresa.

As abordagens gradativas oferecem menos riscos e melhor recuperação de erro. Se um erro ocorrer dentro do módulo migrado, o projeto inteiro não será colocado em risco. Encontrar um erro dentro de um componente migrado é mais fácil do que detectar um erro em algum lugar da aplicação inteira, como pode ocorrer nas abordagens não-gradativas.

Outro fator analisado é o número de pessoas ou equipe envolvidas na realização do processo de modernização, que é uma variável importante a ser analisada, pois está relacionada ao encarecimento do projeto de modernização. Como nas abordagens gradativas a aplicação é migrada aos poucos, por módulos, não é necessário envolver uma grande equipe no processo de migração, o que não acontece em abordagens de migração não-gradativas, que para tentar reduzir o tempo de entrega de resultados precisam alocar uma grande equipe.

As abordagens gradativas oferecem a convivência entre as partes já migradas e as partes legadas da aplicação. Com isso, se reduz o impacto que uma mudança de aplicação traz aos seus usuários, pois, conforme os módulos são migrados, estes são disponibilizados e "agregados", através do interfaceamento com a aplicação legada, facilitando, assim a adaptação à nova aplicação.

Um ponto a ser considerado nas abordagens gradativas é a importância de um bom planejamento e controle do processo, visto que não é uma tarefa fácil. Portanto, ferramentas de apoio são necessárias para facilitar e auxiliar nas tarefas.

Assim, baseado nos atributos discutidos anteriormente conclui-se que uma abordagem ideal deve permitir que a aplicação legada seja migrada de forma gradativa e tenha o máximo de automação no processo, oferecendo maior controle do processo ao Engenheiro de *Software*.

Devido ao fato da abordagem gradativa permitir a modernização por módulos da aplicação, esta deve fornecer diretivas para decompor a aplicação legada em módulos.

Para decompor a aplicação em módulos devem ser identificadas as dependências entre os módulos. Entretanto, esta é uma tarefa difícil, pois está relacionada com o tamanho da aplicação e a qualidade do código.

Decompostos os módulos da aplicação, é necessário apontar qual a seqüência de migração destes módulos, ou seja, priorizar os módulos a serem migrados. Este requisito depende de fatores externos ao processo, pois está voltado às necessidades e prioridades da empresa. Uma possível solução seria apontar alguns critérios a serem analisados como, por exemplo, uma funcionalidade crítica da aplicação, e migrá-la primeiro. Portanto, é necessário fornecer esta flexibilidade de planejamento e controle do processo de migração.

A abordagem ideal também deve oferecer flexibilidade para que alguns módulos sejam totalmente reescritos (ex. nova interface de entrada de dados via *Web*) e outros módulos sejam integrados com a aplicação legada (ex. rotinas para o processamento em modo *"batch"*). Desta forma, a abordagem ideal também deve oferecer recursos de planejamento de migração, para que o Engenheiro de *Software* possa decidir qual modelo de migração usar, quando usar e como usar.

Após a identificação, definição de prioridade e migração do módulo é necessário integrá-los à aplicação legada.. Para tal, deve-se analisar os recursos disponíveis existentes para possibilitar a comunicação entre a aplicação legada, escrita na linguagem origem, com os módulos já migrados, escritos na linguagem destino. Este passo é necessário visto que a comunicação entre aplicações escritas em linguagens diferentes não é uma tarefa trivial. Além disto, não se deve partir do princípio que a tecnologia que está sendo utilizada durante a migração será a tecnologia definitiva, isto porque a escolha da tecnologia a ser utilizada durante o processo visa facilitar a integração com a aplicação legada, enquanto que no resultado final a tecnologia pode ser outra mais nova, robusta e performática.

Devido à complexidade da decomposição dos módulos, de definição da prioridade de migração, de planejamento e controle do processo de migração e a realização da migração de cada módulo, discutidos anteriormente, faz-se necessário o apoio de ferramentas que sejam integradas ou fáceis de integrar ao processo.

Conclui-se então que uma abordagem gradativa deve oferecer os seguintes requisitos:

- Decomposição dos módulos: fornecer diretivas para decompor a aplicação legada em módulos;
- Priorização de migração dos módulos: definir a seqüência de migração dos módulos, obedecendo algum critério;
- Comunicação entre os módulos: especificar a forma de comunicação entre a aplicação legada e a nova aplicação;
- Integração entre ferramentas: fornecer ferramentas que integradas suportem o processo de modernização de aplicações, auxiliando o Engenheiro de *Software* em suas tarefas;
- Tratamento às aplicações procedurais e OO: as abordagens devem tratar os dois tipos de aplicações, uma vez que aplicações OO também precisam ser modernizadas;
- Recuperação de modelos em alto nível de abstração: os modelos ajudam no entendimento e na manutenção da aplicação legada;

2.5 A Integração e o Acesso aos Dados

A integração e o acesso aos dados são fatores importantes e vêm sendo estudados e melhorados, uma vez que as organizações necessitam prover a integração de seus legados e o compartilhamento de seus dados com suas outras aplicações desenvolvidas ao longo dos anos ou mesmo aplicações de terceiros.

No contexto da modernização de aplicações, a integração e o acesso aos dados são essenciais, uma vez que as aplicações devem ser migradas gradativamente e devem permanecer funcionais ao longo do processo.

Assim, é necessário considerar como as organizações têm tratado o compartilhamento de seus dados. Para melhor apresentar algumas das formas utilizadas pelas organizações, vamos dividir os cenários em dois tipos de integração, apresentando as possíveis formas de acesso às informações das aplicações legadas.

2.5.1 Máquina B executa uma nova aplicação e Máquina A executa aplicação legada

Este primeiro cenário é muito comum nos dias de hoje, uma vez que as organizações, normalmente, possuem várias aplicações e precisam que essas se comuniquem entre si ou até mesmo durante o processo de modernização, em que os módulos novos podem acessar a aplicação legada.

A máquina B pode acessar a aplicação legada na máquina A, conforme apresentado na Figura 3 de várias formas:

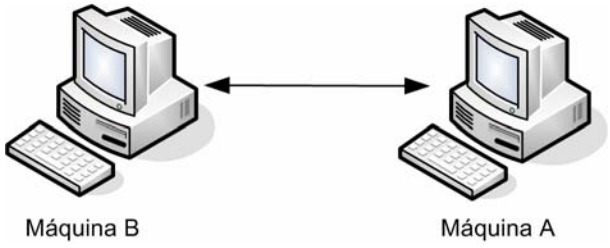


Figura 3 - Formas de Comunicação entre Máquina A e Máquina B

1. Chamada de procedimento remoto (RPC) das rotinas: a chamada de procedimento remoto ou RPC é um tipo de protocolo que possibilita uma aplicação em um computador executar uma aplicação em um servidor.

A aplicação cliente chama uma função ou uma rotina do servidor com os parâmetros desejados, através do envio de uma mensagem e o servidor retorna uma mensagem contendo a resposta da execução.

Desta forma, a nova aplicação, ou seja, a aplicação cliente (máquina B) poderá acessar as rotinas da aplicação legada, ou seja, do servidor (máquina A), conforme apresentado na Figura 4.

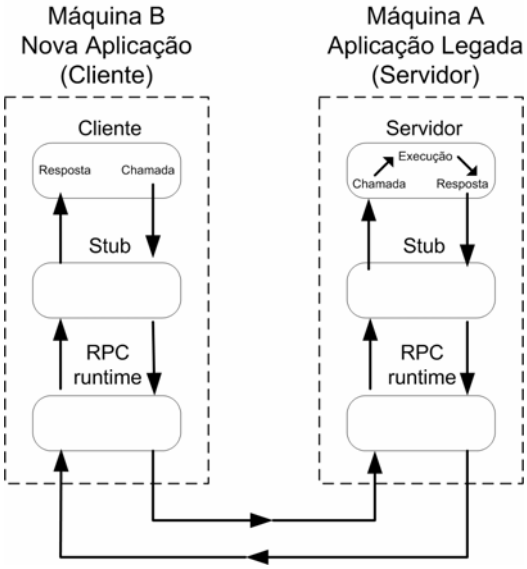


Figura 4 - Implementação do modelo RPC

A principal característica deste modelo é a transparência do serviço, onde o cliente faz a chamada ao procedimento como a um processo local. Esta transparência é possível através dos *Stubs*, que ao receber uma chamada, empacotam os parâmetros do procedimento no formato da

rede e enviam para o RPC *runtime* para enviar ao servidor. Ao receber os resultados, o *Stub* cliente desempacota os dados e os entrega ao processo cliente. O *Stub* do servidor implementa a interface do serviço operando de forma similar ao *stub* do cliente.

O RPC *runtime* implementa a comunicação via rede. No lado cliente, recebe a chamada e encaminha ao RPC *runtime* do servidor. Ao receber a resposta via rede ele encaminha ao *Stub* cliente para o desempacotamento e entrega.

Diferentes tecnologias implementam o modelo RPC:

- CORBA (Common Object Request Broker Architecture) - padrão RPC independente de plataforma.
- Sun RPC (Remote Procedure Call) - RPC para as plafaformas Unix e Linux
- DCE (Distributed Computing Environment)
- DCOM (Distributed Component Object Model) - RPC para plataforma Windows.
- RMI (Remote Method Invocation) - RPC para Java.
- SOAP (Simple Object Access Protocol) - padrão de RPC para *Webservices*.
- JCA (Java Connector Architecture)
- COMTI (COM Transaction Integration)

2. **Wrapper**: é um código combinado com um outro pedaço de código que determina como o código será executado. O *wrapper* age como uma interface entre o seu chamador e o código envolvido pelo *wrapper*. Um objeto *wrapper* é uma transformação caixa preta de código legado, que fornece uma interface para uma aplicação ou componente.

Esta técnica pode ser usada para resolver questões de compatibilidade quando, por exemplo, o código envolvido está escrito em uma linguagem diferente ou faz chamadas de formas diferentes, ou ainda por questões de segurança, por exemplo, para previnir o programa chamado de executar certas funções. O código envolvido só pode ser acessado pelo *wrapper*.

O *wrapper* envolve os dados, aplicações e interfaces com novas interfaces. A camada *wrapping* pode se comunicar com a aplicação legada através de *sockets*, RPCs, (chamada de procedimento remoto), ou pela interface de uma aplicação. O *wrapper* expõe uma interface baseada em objeto a uma aplicação legada, que não precisa ser alterada, escondendo as telas, APIs, adaptadores de comunicações, arquivos, e bancos de dados.

Depois de envolvida pelo *wrapper*, a aplicação legada se torna um componente de *software* reutilizável. O resultado é que as aplicações legadas podem ser invocadas por novas aplicações, estendendo e aumentando assim sua vida e a de seus dados.

Uma das implementações de *wrapping* mais utilizada é o *screen-scraping*, apresentada a seguir.

3. **Screen-Scraping**: esta técnica consiste em envolver as interfaces legadas baseadas em caractere com novas interfaces gráficas.

Desta forma, levando em consideração este primeiro cenário, a nova aplicação executada na máquina B poderá acessar a aplicação legada executada na máquina A através de um *screen-scraper* que vai envolver a aplicação legada com uma nova interface, conforme apresentado na Figura 5.

O *screen-scraper* lê o fluxo de dados enviado pela aplicação cliente (máquina B) para a aplicação legada na máquina A, e o transforma em uma apresentação gráfica mais atraente que as baseadas em caractere. Esta representação pode ser uma interface gráfica (GUI) e até mesmo uma interface HTML que é executada em um navegador *Web*.

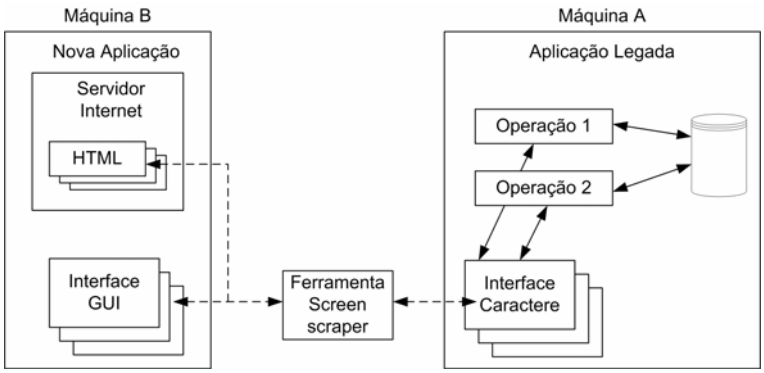


Figura 5 - Técnica de Screen scraping

Esta técnica é um dos métodos mais bem sucedidos para extensão de aplicações mainframe para melhorar e estender a funcionalidade das aplicações legadas. As ferramentas de *screen-scraper* evitam alterações nas aplicações mainframe, tornando-as mais fáceis de usar. Elas também permitem modificações extensas na seqüência de informações apresentada ao usuário, combinando várias telas em uma única apresentação gráfica, dando o efeito de modificação na aplicação sem alterar a lógica de negócio da aplicação.

Um problema dessa técnica é que se alguma alteração for feita no código mainframe, há grandes possibilidades do *screen-scraper* parar de funcionar.

4. **Transferência de Arquivos**: esta técnica visa a transferência do arquivo de dados de uma aplicação para outra em uma LAN.

O problema com esta técnica é que ela faz uso de grande quantidade de largura de banda, especialmente para dados acessados com mais freqüência. Além disso, tempo e recursos são

desperdiçados transferindo arquivos inteiros quando o usuário só precisa de alguns pedaços da informação. Duas ferramentas se destacam na transferência de arquivos:

- MQSeries (IBM);
- MSMQ (Microsoft);

Essas duas ferramentas podem ser executadas de forma síncrona ou assíncrona. Elas recebem um arquivo ou até mesmo um texto e enviam para outro lugar, e garantem que vai ser mesmo entregue do outro lado, mesmo que for de forma assíncrona. Isto também é bastante usado, principalmente para transferência assíncrona.

2.5.2 Máquina C executa aplicação reescrita na baixa plataforma (Windows) e acessa o banco de dados remoto

Este cenário reflete a necessidade de integração da aplicação que acabou de ser reescrita na baixa plataforma (Windows) e ainda faz acesso ao banco de dados remoto.

Existem duas formas para realizar este acesso:

1. ODBC *(Open Database Connectivity):* é uma especificação de interface para acesso a dados. Essa especificação provê funções para conectar e desconectar fontes de dados. Permite que uma aplicação acesse simultaneamente uma variedade de SGBD's, (Sistema de Gerenciamento de Banco de Dados) relacionais e não-relacionais, situados em locais diferentes, através de uma única API *(Application Programming Interface).*

O objetivo do ODBC é tornar possível o acesso a qualquer dado de qualquer aplicação, independentemente do SGBD que esteja gerenciando os dados. O ODBC gerencia o acesso inserindo uma camada, chamada de *driver* de banco, entre a aplicação e o SGBD. O propósito desta camada é traduzir as consultas dos dados da aplicação em comandos que o SGBD conheça. Para isso, tanto a aplicação como o SGBD precisam ser compatíveis com o ODBC, ou seja, a aplicação deve ser capaz de emitir comandos ODBC e o SGBD deve ser capaz de respondê-los.

Conforme apresentado na Figura 6, a nova aplicação é executada na máquina C na baixa plataforma acessando o banco de dados remoto através da API ODBC.

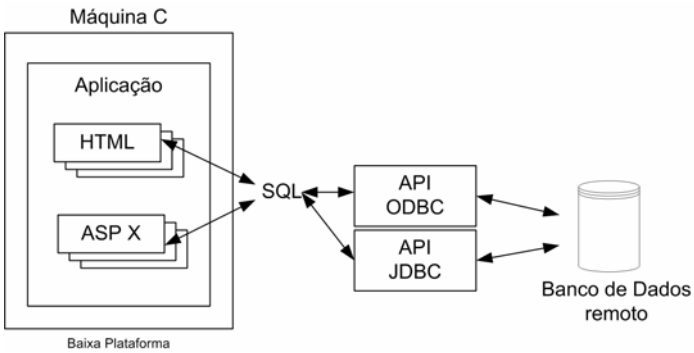


Figura 6 - Acesso a banco de dados remoto através do ODBC

2. JDBC *(Java Database Connectivity)*: é uma API destinada à integração da linguagem Java a banco de dados relacionais. Esta API define classes Java para representar e operar conexões com bancos de dados, cujo objetivo é prover uma interface independente de qualquer RDBMS (Sistema de Gerenciamento de Bancos de Dados Relacional), permitindo o acesso genérico a bancos de dados através de SQL de modo uniforme, mesmo para diferentes fontes de dados.

Conforme apresentado na Figura 7, a aplicação C é executada na baixa plataforma acessando o banco de dados remoto através da API JDBC.

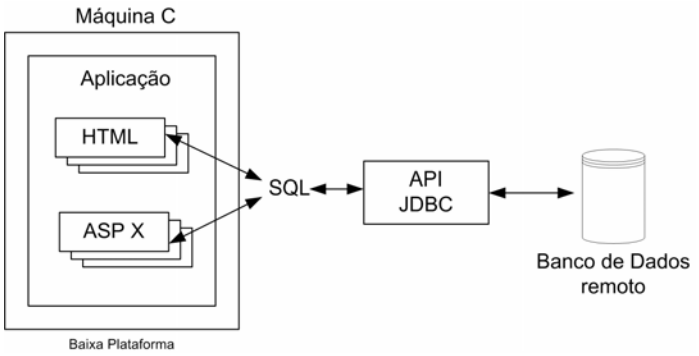


Figura 7 - Acesso ao banco de dados remoto através do JDBC

3. HIS: *(Host Integration Server)*: permite o acesso e integração de aplicações críticas, fonte de dados e sistemas de segurança baseados no *hosts* IBM com as novas aplicações desenvolvidas na plataforma Windows. Isto possibilita o reuso dos dados dos mainframes IBM e de aplicações em ambientes distribuídos. O *Host Integration Server* possibilita a integração com banco de dados relacionais, através do ODBC e do OLE DB *Provider,* e não-relacionais através do OLE DB *Provider* para arquivos. Conforme apresentado na Figura 8, a aplicação C é executada na baixa plataforma acessando o banco de dados remoto DB2 e os arquivos VSAM do *mainframe*.

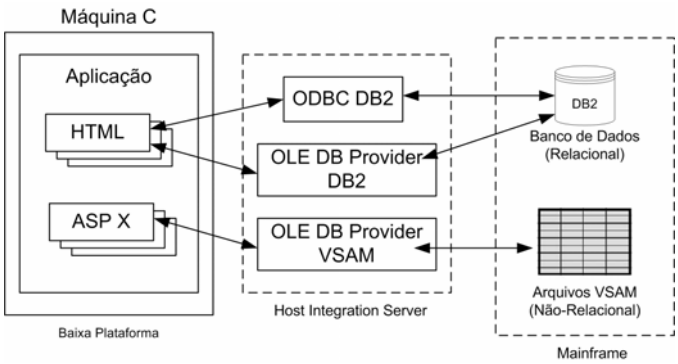


Figura 8 - Acesso ao banco de dados remoto através do HIS

2.6 Possíveis Cenários na Migração Gradativa

Em grandes empresas, existe um grande número de aplicações legadas que estão em constante manutenção há anos. A migração destas aplicações torna-se praticamente inviável de ser realizada de uma só vez, pois este processo pode levar anos. Desta forma, as empresas optam por migrar aos poucos a aplicação, mantendo a nova aplicação em comunicação com a aplicação legada.

Para melhor entender a idéia deste processo, serão apresentadas algumas possíveis situações onde a migração gradativa pode ocorrer e como ela pode ser aplicada. O objetivo é demonstrar, com exemplos, a utilização das técnicas apresentadas.

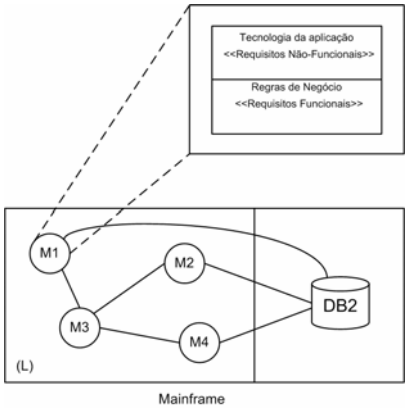


Figura 9 - Aplicação Legada (L)

Na Figura 9 é apresentada a aplicação legada (L) de uma empresa, por exemplo, escrita em COBOL, executada em um mainframe, composta dos módulos M1, M2, M3 e M4 acessando uma base DB2, onde cada módulo é composto por requisitos não-funcionais e requisitos funcionais.

Considerando um caso em que a empresa deseja converter esta aplicação, seguem alguns dos possíveis cenários para exemplificar o processo.

2.6.1 Cenário 1 – Migrar as Interfaces de Usuário dos Módulos para Web, mantendo inalteradas as Regras de Negócio e a Base de Dados Legada

Considerando que a empresa possua uma aplicação escrita em COBOL, com as interfaces procedurais dos módulos de Vendas (M1, M2 e M3), a serem reconstruídas em páginas *Web*.

Na Figura 10 tem-se uma possível estratégia para atender os requisitos do Cenário 1, composta de 3 passos:

- Passo 1: migrar para *Web* as interfaces do usuário dos módulos M1, M2 e M3 (1);
- Passo 2: estabelecer a comunicação entre as novas interfaces (Páginas *Web*) e as regras de negócio legadas não-convertidas do *mainframe*. A comunicação pode ser realizada através de componentes *Web Services* (2a) ou ainda através de um objeto *wrapper* (2b);

Em (2a) as funcionalidades das regras de negócio são disponibilizadas como serviços pelos componentes *Web Services*.

Em (2b) o objeto *wrapper* disponibiliza uma interface para acesso às funcionalidades das regras de negócio da aplicação legada.

- Passo 3: manter as regras de negócio legadas acessando diretamente a base de dados legada, no caso DB2 (3);

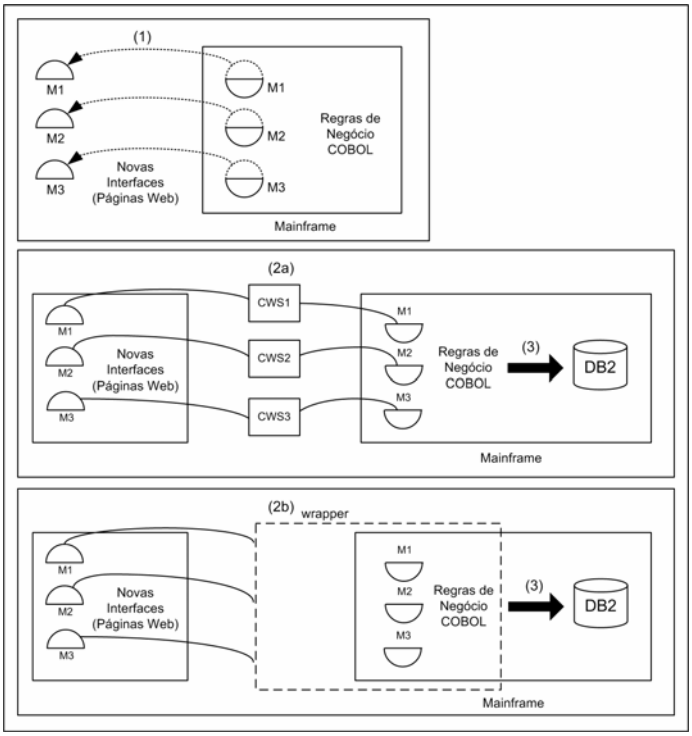


Figura 10 - Possível estratégia para o Cenário 1

2.6.2 Cenário 2 – Migrar as Interfaces de Usuário e as Regras de Negócio do Módulo M4 para .NET e Java, mantendo o Acesso à Base de Dados Legada

Considere que o módulo M4 seja referente às funções financeiras, e a empresa vende este módulo para vários clientes. Para aumentar seus negócios, a empresa precisa atender às necessidades de cada cliente. Como nem todo cliente trabalha com a linguagem e plataforma na qual o produto foi construído, a empresa fica impossibilitada de vender seu produto a estes clientes, já que um cliente que só trabalha com Java ou .NET não vai comprar um produto desenvolvido em COBOL.

Portanto, a empresa decide migrar o seu produto para as linguagens .NET e Java para atender a estes clientes.

Na Figura 11 tem-se uma possível estratégia, composta de 3 passos, para atender os requisitos do Cenário 2, migrando o módulo M4 para estas novas linguagens.

- Passo 1: migrar as interfaces de usuário do módulo M4 para .NET e Java (1);
- Passo 2: migrar as regras de negócio do módulo M4 para .NET e Java (2);
- Passo 3: estabelecer a comunicação do novo módulo M4 em .NET e Java com a base de dados legada, DB2 (3). Esta comunicação pode ser realizada em .NET através da API *OLE DB*

Provider para DB2 e em Java através de *JDBC* ou ainda por uma API para comunicação de aplicações Java e base de dados DB2 fornecida por terceiros.

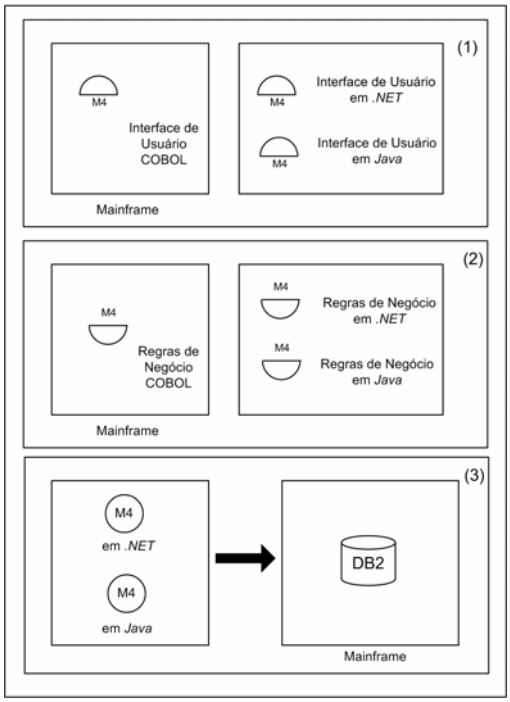


Figura 11 - Possível estratégia para o Cenário 2

2.6.3 Cenário 3 – Alterar a Base de Dados Legada (DB2) para Oracle

Considerando que a empresa deseja converter sua aplicação para uma nova plataforma, por exemplo, Unix/PC, esta deverá converter sua base de dados, a menos, por exemplo, que aceite comprar uma versão de sua base de dados, DB2, para a nova plataforma.

Na Figura 12 tem-se uma possível estratégia para o Cenário 3, composta de 3 passos.

- Passo 1: exportar os dados da base legada DB2, via DDL *(Data Definition Language)* [ELMASRI; NAVATHE 2000], através de alguma ferramenta de apoio (1);
- Passo 2: importar a DDL na nova base de dados Oracle (2);
- Passo 3: estabelecer a comunicação entre a aplicação e a nova base de dados, através de *JDBC* no caso de uma aplicação Java ou Ado.Net, no caso de uma aplicação .NET (3).

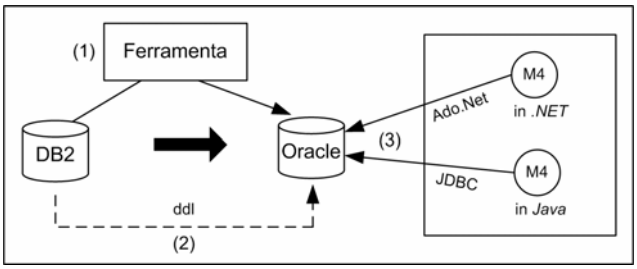


Figura 12 - Possível estratégia para o Cenário 3

2.6.4 Cenário 4 – Aumentar a Satisfação do Usuário quanto à Aparência das Interfaces Caractere da Aplicação Legada

Considerando a necessidade de oferecer uma interface mais atraente ao usuário da aplicação legada, uma solução mais rápida e também eficiente seria o uso de *screen-scraper* para simular uma interface mais atraente aos usuários da aplicação legada.

Na Figura 13 tem-se uma possível estratégia para o Cenário 4, composta de 1 passo.

- Passo1: criar *screen-scraper* para as interfaces da aplicação legada;

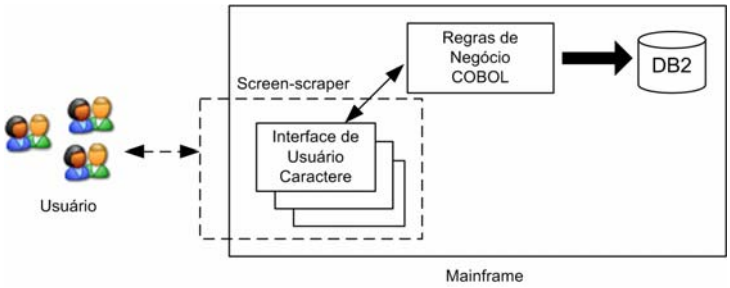


Figura 13 - Possível estratégia para o Cenário 4

Utilizando o *screen-scraper* não há necessidade de alterações no código legado da aplicação. Estes são somente alguns exemplos da flexibilidade usando uma migração gradativa.

2.7 Ferramentas de Apoio

2.7.1 Sistema de Transformação Draco-PUC

O Draco é um sistema transformacional genérico construído com o objetivo de testar, desenvolver e colocar em prática o paradigma Draco para o desenvolvimento de *software*. É baseado em um Motor Transformacional (MT) para manipulação de Árvores de Sintaxe Abstratas Draco (*DAST - Draco Abstract Syntax Trees*). Um primeiro protótipo do ST Draco foi construído por Neighbors [NEIGHBORS 1984]. Posteriormente, o ST Draco foi reconstruído na PUC-RJ [LEITE et al. 1994] e [LEITE et al.1996], usando novas linguagens e plataformas de *hardware* e *software,* tornando-se mais poderoso. O ST Draco-PUC passou ainda por duas novas atualizações em 1997 e 2001.

A geração automática de código, no ST Draco-PUC, baseia-se em transformações, orientadas a domínios, que mapeiam a sintaxe e semântica dos comandos de um programa, de um domínio, em outro programa, do mesmo ou outro domínio. O ST Draco-PUC possui:

- Um Sistema gerador de analisadores léxicos e sintáticos (parser) baseado na sintaxe Lex/Yacc, utilizando mecanismo de análise LALR (*Look-Ahead Left Right*) com *backtracking*;

- Um núcleo transformacional totalmente aberto, que suporta o uso de pontos de controle para o disparo de eventos e direção de fluxo de controle. Mecanismos de controle garantem a exatidão da execução das transformações;
- Uma linguagem para descrição das transformações, a qual usa transformações locais e globais;
- Mecanismo de casamento de padrões altamente expressivo, fornecendo ao usuário um grande potencial para as especificações dos requisitos de *software*, utilizando a sintaxe de qualquer linguagem que tenha sido definida no subsistema de *parsers*;
- Facilidade para agrupar transformações em conjuntos com diferentes pontos de controle, e;
- Mecanismo para aplicação dos componentes transformacionais que mapeiam a sintaxe e semântica de uma especificação de entrada para uma nova especificação do mesmo ou de outro domínio. Por exemplo, para geração automática de código em linguagem de programação como C++, Pascal e Java, a partir de um código legado em Clipper, ou de uma especificação em uma linguagem de modelagem.

Um domínio no ST Draco-PUC é definido através de uma Linguagem; esta por sua vez é formada por uma Gramática, um *Parser* e *Prettyprinter*, ou *unparser*, definidos a partir desta gramática, conforme apresentado na Figura 14.

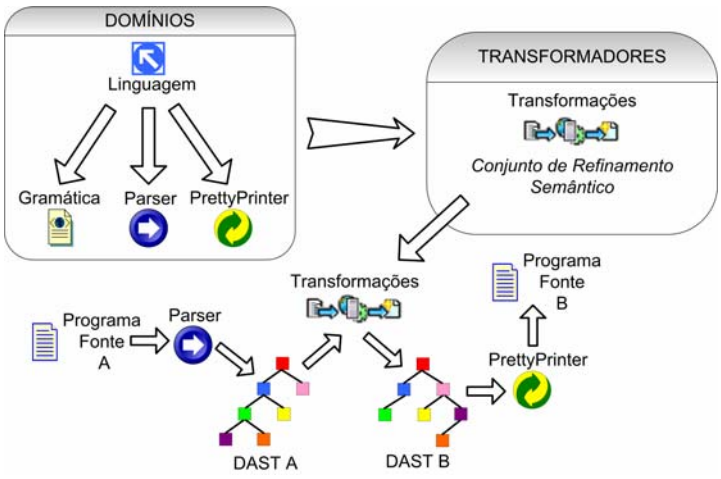


Figura 14 – Domínio no ST Draco-PUC

Uma Linguagem pode ser definida como qualquer conjunto de sentenças geradas a partir de um conjunto finito de símbolos. É formada por:

- Gramática: consiste em terminais, não-terminais, um símbolo de partida e produções. Os terminais, ou *tokens*, são símbolos básicos a partir dos quais as cadeias são formadas. Os não-terminais são variáveis sintáticas que denotam cadeias de caracteres. Os não-terminais definem conjuntos de cadeias que auxiliam a definição da linguagem gerada pela gramática. Um símbolo de partida, numa gramática, é um não-terminal, e o conjunto que ele denota é a linguagem definida pela gramática. As produções de

uma gramática especificam a forma pela qual os terminais e não-terminais são combinados para formar cadeias.

- *Parser:* gerado, automaticamente, a partir das definições da gramática e do Analisador Léxico da linguagem do domínio, pelo componente PARGEN do ST Draco-PUC. O *parser* é responsável por analisar um programa qualquer deste domínio e gerar, automaticamente, sua representação interna no ST Draco-PUC. Nesta representação interna, que no ST Draco-PUC é denominada Draco Syntax Abstract Tree (DAST), são aplicadas as transformações, que geram uma nova DAST no mesmo ou em outro domínio.
- *Prettyprinter:* ou *unparser,* é responsável por escrever representações de uma DAST, tornando-a novamente textual na linguagem do domínio. Baseado nas definições das gramáticas o subsistema PPGEN do ST Draco-PUC gera, automaticamente, os respectivos *prettyprinter* dos domínios.

Conforme apresentado na Figura 14, inicialmente, parte-se de um Programa Fonte A, que é analisado pelo *parser* do seu domínio, gerando sua representação interna DASTA. As transformações são aplicadas nessa DASTA, gerando uma nova DASTB, que submetida ao *Prettyprinter* do domínio, é reescrita na forma textual da linguagem do domínio, obtendo o Programa Fonte B.

O nível de abstração da linguagem de um domínio pode ser tanto o de implementação, quanto o de modelagem ou especificação mais próximo do usuário. Esta é uma das características que o ST Draco-PUC oferece ao Engenheiro de *Software*, e que facilita obter o projeto da aplicação, a partir do seu código fonte, representando-o com técnicas de modelagem, em alto nível de abstração.

Os Transformadores são pacotes de transformações que atuam numa DAST, para gerar uma nova DAST. Existem dois tipos de transformações, definidas nos transformadores. São elas:
Intra-Domínio: corresponde às transformações horizontais, e mapeiam estruturas de uma linguagem para estruturas na mesma linguagem do domínio. Normalmente são utilizadas para otimização ou organização de programas, e;
Inter-Domínios: correspondem às transformações verticais, e mapeiam as aplicações descritas em uma linguagem de um domínio para descrições de linguagem de outro domínio. Essas promovem a mudança de domínio.

As transformações são responsáveis pela automatização ou semi-automatização do processo de construção de *software*, através da otimização e manipulação da DAST. Uma transformação deve ser composta basicamente de um padrão de reconhecimento, chamado LHS (*Left-Hand-*

Side) e um padrão de substituição, chamado de RHS (*Right-Hand-Side*), conforme apresentado na Figura 15.

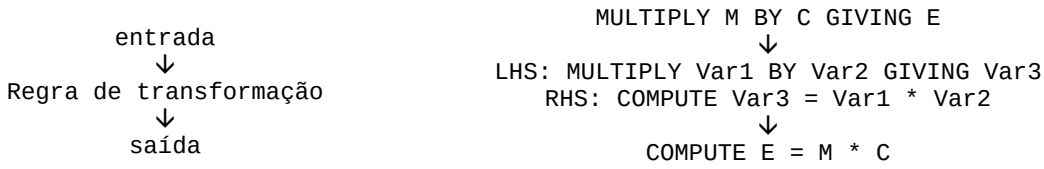


Figura 15 – Exemplo de Transformação

As transformações são armazenadas em transformadores organizados de acordo com o *Framework* da Figura 16. Cada transformador *(Transformer)* pode conter seções globais *(Global-Declaration, Global-Initialization* e *Global-End)* e um ou vários conjuntos de transformações *(Sets of Transforms).* Cada *Set of Transforms* é formado por várias transformações *(Transforms).* O Engenheiro de *Software* pode aplicar as transformações em diferentes níveis de granularidade, desde um simples *token* até um grande programa.

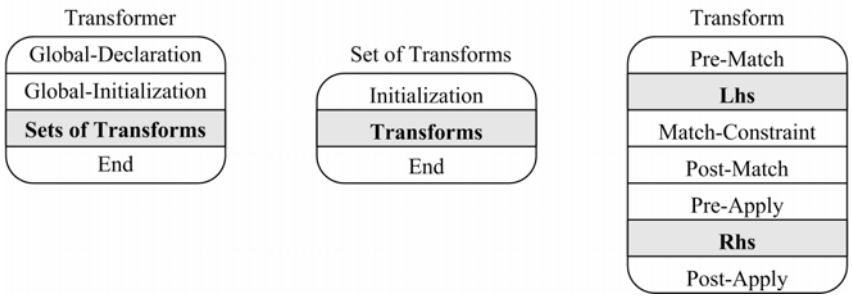


Figura 16 – Framework para as Transformações do Draco-PUC

Conforme o esquema apresentado, um transformador contém as seguintes seções:

- **Global Initialization**: área reservada para as inicializações globais. É executado quando o transformador for selecionado para a aplicação;
- **Global Declaration**: área reservada para declaração de variáveis globais; e
- **End**: executado quando o transformador terminar de ser aplicado.

O *Set of Transforms*, é um conjunto de transformações, que possuem os seguintes pontos de controle:

- **Initialization**: executado sempre que o conjunto de transformações for selecionado para aplicações; e
- **End**: executado sempre que o conjunto de transformações terminar de ser aplicado.

Cada *Transform* dispõe além dos LHS e RHS, dos seguintes pontos de controle:

- **Pre-Match**: executado sempre que a regra de transformação é testada sobre um trecho da descrição de entrada;
- **Post-Match**: executado após a regra de transformação ser testada sobre um trecho da descrição de entrada;
- **On-Match-Exit**: executado logo após ter sido finalizada uma tentativa de unificação entre o lado esquerdo e um trecho da descrição de entrada;
- **Pre-Apply**: executado imediatamente antes da substituição do trecho de entrada selecionado pelo lado direito, e
- **Post-Apply**: executado após a substituição do trecho de entrada selecionado pelo lado direito.

Os conjuntos de transformações também permitem especificar como as suas transformações serão aplicadas na DAST do programa alvo das transformações, de cima para baixo *(Top-Down),* de baixo para cima *(Bottom-Up)* ou seqüencial *(Sequential).* A aplicação das transformações pode ser em um único passo *(Single Step),* para transformações Inter-Domínios, ou em vários passos *(Exhaustive)* para transformações Intra-Domínio.

Eventualmente, uma transformação necessita de informações capturadas por outras transformações. Se essa informação for composta de muitas linhas de código, áreas de trabalho chamadas de *WORKSPACES* podem ser usadas. A estratégia de transformação por *WORKSPACES*, por exemplo, utiliza os pontos de controle para adiar a reescrita de uma regra para fazer uma difusão ou concentração de uma descrição de uma linguagem para outra. Assim, uma transformação, desprovida de LHS, pode passar meta variáveis através de um *TEMPLATE*, que é um padrão formatado de uma dada categoria sintática, armazenando-o em um *WORKSPACE*. O *TEMPLATE* armazenado pode então ser reutilizado por outras transformações.

Caso necessário, nos pontos de controle, o Engenheiro de *Software* pode adicionar ações semânticas, escritas na linguagem C++. Dessa forma, além dos *WORKSPACES*, pode-se contar com todos os recursos da linguagem C++, que permite resolver problemas específicos não previstos no *Framework* das transformações.

Para facilitar o tratamento de informações semânticas, o ST Draco-PUC dispõe de um recurso de Base de Conhecimento *(KB - Knowledge Base),* que captura informações sobre o programa do domínio analisado, e libera o Engenheiro de *Software*, em alguns casos, do uso de estruturas de dados auxiliares. Esse recurso armazena cláusulas, fatos e regras, em uma notação similar a Prolog, que serão consultadas posteriormente. Comandos utilizados nos pontos de controle dos

componentes permitem armazenar, consultar, recuperar e excluir fatos ou regras na Base de Conhecimento, como por exemplo, KBAssert(), KBAssertIfNew(), KBSolve(), KBRetrieve() e KBDelete(), respectivamente.

A construção das transformações dispõe de outros recursos, que permitem a comunicação de funções e macros do ST Draco-PUC com a linguagem C++ e vice-versa, uma vez que o transformador é construído em C++. Um exemplo é a função *expand* que exporta uma metavariável Draco como *String* para ser usada como uma variável C++ num ponto de controle, e a função *Set_Leaf_Value* que permite associar uma metavariável Draco a uma *String* com valor pré-definido armazenado em uma variável C++.

Uma transformação pode também, por meio do comando *Apply*, chamar outra transformação ou um conjunto de transformações passando metavariáveis como parâmetro.

O ST Draco-PUC é atuante nas pesquisas realizadas para a reengenharia de aplicações legadas. Seus transformadores vêm sendo empregados na conversão de aplicações nos domínios de linguagem de programa como Progress para Java [NOVAIS 2002], Dataflex para Visual Dataflex [NOGUEIRA 2002], Cobol para C++ [LEITE et al. 1996] e Clipper para Java [JESUS 2000]. Também têm-se experiências em domínios de modelagem, como MDL *(Modeling Domain Language)* do projeto RST [FONTANETTE et al. 2001].

2.7.2 Apyon Studio

O Apyon Studio [OLIVEIRA 1998], [OLIVEIRA; PALAZZO 1999], [APYON 2000] é o principal produto da Apyon Technology S/A. O Apyon Studio é uma ferramenta de produtividade que trabalha em alto nível de abstração, sem considerar a complexidade tecnológica. O Apyon Studio é utilizado na etapa de projeto pelos analistas de sistemas, através de integrações com Ferramentas CASE existentes no mercado, como o Microsoft Viso, ERWin, Oracle Design, Rational Rose, System Architect e Power Design.

Após a especificação da aplicação utilizando o Apyon Studio, o conhecimento do negócio fica armazenado em seu repositório. O Apyon Studio dispõe de mecanismos e *templates* de geração de código baseado nas informações do repositório, sendo utilizado também na etapa de implementação. Os *templates*, que utilizam o conceito de *design patterns*, convertem as especificações abstratas do repositório em código fonte em diversas tecnologias. O Apyon Studio viabiliza o rápido *Design*, Desenvolvimento, *Deploy* e Manutenção de complexas aplicações *e-business*, tornando-as portáveis para as principais tecnologias, reduzindo custos, prazos e riscos.

Sua característica principal é a abstração da complexidade tecnológica e conseqüentemente eliminação dos riscos relacionados com a tecnologia empregada nos projetos. O uso do Apyon

Studio faz com que sejam retiradas as preocupações com tecnologia do processo de desenvolvimento e faz com que o Engenheiro de *Software* e sua equipe trabalhem focados nas funcionalidades e requisitos de negócio. O Engenheiro de *Software* fornece informações para o produto, que são transformadas em código fonte puro no padrão definido pelo cliente. O programador recebe tarefas de programação (regras de negócio) que, unidas ao código gerado automaticamente, formam a aplicação final.

Os padrões de geração de código do Apyon Studio implementam o conceito de melhores práticas tecnológicas e podem ser facilmente adaptados para implementar os padrões específicos de cada organização. Isto garante que as aplicações desenvolvidas pelos engenheiros de *software* estejam sempre alinhadas com os padrões tecnológicos definidos pela área de arquitetura e padrões de cada empresa. Se uma nova versão de uma tecnologia Java, por exemplo, chegar ao mercado ou se o Engenheiro de *Software* descobrir uma maneira melhor de, por exemplo, controlar segurança, estes padrões tecnológicos podem ser alterados para fazer uso das melhorias implementadas na tecnologia. Assim todas as aplicações que foram desenvolvidas com o Apyon Studio podem ser geradas novamente para fazerem uso destas melhorias. Com isto as aplicações não ficam velhas ou ultrapassadas.

O primeiro passo para a utilização do Apyon Studio é a importação das informações de modelo estático de uma ferramenta de análise e modelagem para seu repositório. Uma vez realizada a importação do modelo estático, a próxima etapa do processo é a especificação funcional das interfaces de usuário e das regras de negócio. As interfaces são especificadas com base nas classes persistentes ou nas tabelas e seus relacionamentos. Toda especificação é realizada em alto nível, sem necessidade de detalhar aspectos sobre tecnologia ou *design*.

Visando garantir um processo consistente e padronizado de especificação, o Engenheiro de *Software* é conduzido através de *wizards*, que sugerem e validam informações automaticamente, reduzindo a possibilidade de especificações inconsistentes.

Para a especificação das interfaces de usuário é utilizado o módulo "*User Interface Manager*" apresentado na Figura 17.

Este módulo permite especificar os Menus da aplicação, os filtros de consulta

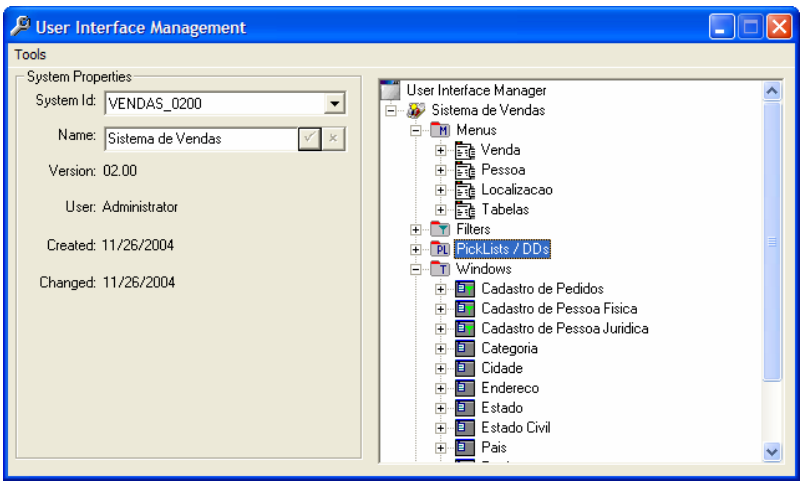


Figura 17 – User Interface Manager

(informações solicitadas ao usuário para restringir a seleção em uma base de dados), *pick-list/drop downs* (lista de seleção de informações), *windows* (interfaces de usuário).

Na Figura 18 é mostrado um exemplo de como é feita à especificação de interface de usuário.

A especificação é realizada em alto nível, sem citar os aspectos tecnológicos de implementação, pois essas regras serão programadas posteriormente. Na Figura 19 tem-se um exemplo de especificação das regras de negócio.

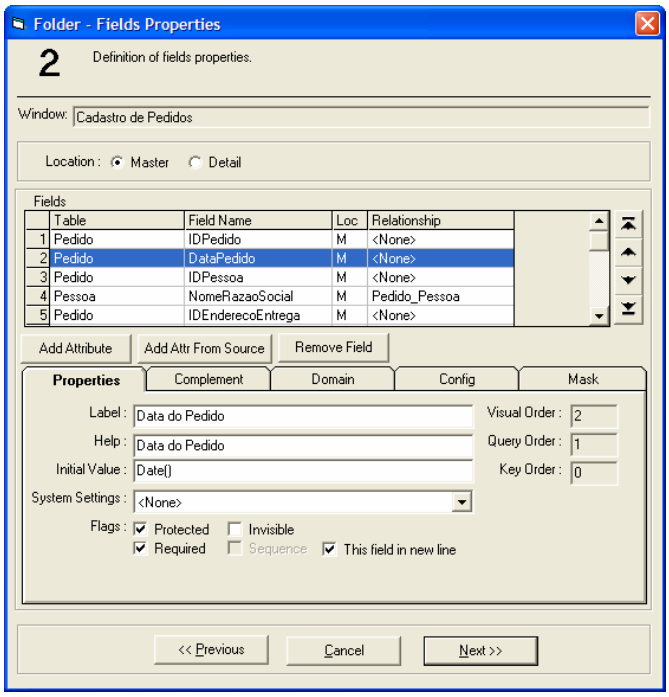


Figura 18 - Definição das propriedades das interfaces

Uma regra de negócio pode estar ligada a uma classe persistente ou pode ser um processo. Se estiver ligada a uma classe, pode também estar ligada a um evento de inclusão, alteração ou exclusão. Desta forma, a regra é executada automaticamente no momento que um destes eventos ocorrer. Se a regra for especificada com o tipo processo, será gerado um método ou um componente genérico que pode ser utilizado em qualquer ponto da aplicação.

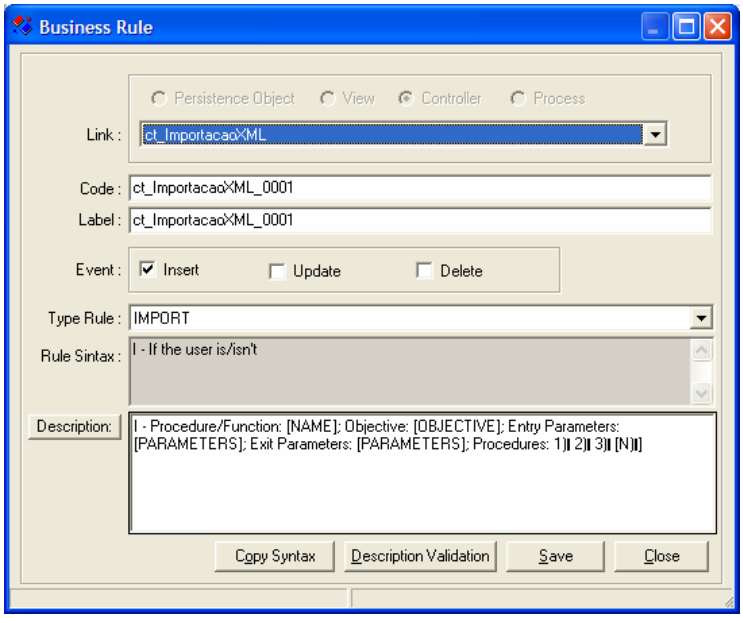


Figura 19 – Especificação de Regra de Negócio

Quando o Engenheiro de *Software* conclui a especificação da regra de negócio, o Apyon Studio gera automaticamente um método para suportar e gerenciar a implementação desta regra. Se a regra for ligada a uma classe persistente, o método é automaticamente adicionado ao componente que implementa a persistência. Neste componente é possível organizar a seqüência de execução de cada regra de negócio e também inserir os parâmetros que serão necessários para a implementação.

Após a especificação das funcionalidades da aplicação no Apyon Studio, é possível gerar automaticamente o código para as tecnologias definidas.

Primeiramente são gerados os componentes de persistência, com as respectivas chamadas às regras de negócio. O Apyon Studio gera toda a infra-estrutura tecnológica, como os comandos de acesso ao banco de dados, os parâmetros utilizados pela regra, a seqüência de execução entre as regras, o gerenciamento de erros e de exceção, recurso de multi-idioma, a conexão com o banco de dados e todo o controle de transação e a documentação do código. Essas funcionalidades geradas automaticamente correspondem a maior parte do código dos componentes e também a parte que mais exige conhecimento tecnológico. Sem o Apyon Studio, a implementação dessas funcionalidades exigiria muito conhecimento tecnológico dos engenheiros de *software* e sua equipe.

Apenas o código da regra de negócio não é gerado e deve ser feito manualmente no local especificado pelo Apyon Studio. Assim, a programação manual fica reduzida a uma porção muito pequena, focada nos aspectos funcionais da regra e abstraído da complexidade tecnológica. Por exemplo, uma regra de validação precisará de apenas uma linha de código, já que toda a parte de tecnologia foi automaticamente gerada. Isto reduz o tempo de implementação, simplificando o processo e aumentando a qualidade por envolver menos codificação manual.

Em seguida são geradas as interfaces de usuário com as funcionalidades especificadas no "*User Interface Manager*". As especificações armazenadas no repositório são geradas em uma tecnologia específica. O código é gerado com funcionalidade de controle de segurança de acesso, gerenciando os *logins* na aplicação, controle de sessão, o menu dinâmico, de acordo com as permissões de grupos e usuários e também as permissões de operação (inclusão, alteração, exclusão e consulta). Todas as interfaces de usuário geradas contam com recursos de multi-idioma, o que significa que todas as informações que são apresentadas aos usuários, incluindo *labels*, mensagens e menus, podem ser facilmente convertidas para outro idioma, sem nenhuma modificação em código fonte.

Neste ponto, os códigos fonte das interfaces de usuários e dos componentes já gerados e compilados estão prontos para serem utilizados.

É possível trocar o banco de dados sem necessidade de fazer nenhuma alteração nos códigos-fonte existentes. Isto é possível porque o Apyon Studio disponibiliza um componente "tradutor" junto com fontes da aplicação gerada. Este componente converte os comandos SQL para o banco de dados desejado antes de submeter os acessos.

Em seguida, é gerado um *kit* de programação que contém informações necessárias para que um programador possa desenvolver as regras de negócio definidas pelo Engenheiro de *Software*.

O *kit* é composto pelo componente onde a regra de negócio será programada, a especificação da regra de negócio, os comandos para a criação do banco de dados (somente das tabelas envolvidas) e os dados de teste.

Os *kits* podem ser distribuídos em forma de tarefas para programadores internos ou externos à empresa. Os programadores que implementarão as regras não necessitam ter acesso ao Apyon Studio. Para programar as regras irão usar o ambiente de desenvolvimento que já estão acostumados.

À medida que as regras de negócio são concluídas e validadas é necessário alimentar o repositório do Apyon Studio com o código programado. Somente o código fonte programado é importado para o repositório.

Como o código fonte programado é mantido no repositório do Apyon Studio, é possível regerar a aplicação a qualquer momento. Se forem necessárias alterações na aplicação, como, por exemplo, incluir um novo atributo em uma classe, a especificação pode ser modificada e o código pode ser regerado, sendo que o código do componente (parâmetros, comandos SQL, etc) receberá o novo atributo automaticamente.

O código de uma regra de negócio é independente do código do componente. Isto significa que toda a parte do código que trata a transação, conexão com banco de dados, fluxo de execução da aplicação e comandos SQL são gerados automaticamente, sem influenciar o código fonte das regras de negócio programado manualmente. Isto permite também a troca da arquitetura da aplicação (por exemplo, uma regra escrita em Java para a arquitetura Cliente e Servidor, pode ser convertida e aproveitada em uma aplicação gerada para a arquitetura J2EE) sem perder o código programado.

O Apyon Studio trabalha com o conceito de Documentação Ativa, por isso é possível gerar no formato HTML a documentação para todos os elementos especificados. As documentações geradas sobre as especificações das interfaces de usuário e das regras de negócio são exatamente as especificações que seriam definidas manualmente pelos engenheiros de *software* durante o projeto da aplicação.

É muito importante que a equipe de tecnologia mantenha a documentação da aplicação e faça o controle e o gerenciamento de versões da aplicação. Para isso, o Engenheiro de *Software* inicia novamente o processo revisando as especificações. Desta forma o processo de desenvolvimento e manutenção de aplicações é executado de forma interativa, passando por todas as etapas sem que isto obrigue um re-trabalho. Isto só é conseguido com uma ferramenta que auxilie nas especificações, gere a aplicação, integre com outras ferramentas e mantenha uma documentação ativa.

A aplicação desenvolvida pelo Apyon Studio não possui qualquer dependência de sua estrutura. Se por algum motivo a empresa quiser dar manutenção diretamente no código final, sem usar o Apyon Studio, poderá fazê-lo sem qualquer restrição. As únicas restrições para manutenção externa referem-se à produtividade, qualidade e custo dessa manutenção.

Possibilitando a abstração da complexidade tecnológica e unindo funcionalidades de geração automática de código e gestão da programação de regras de negócio, o Apyon Studio viabiliza uma redução de mais de 30% nos prazos e custos do desenvolvimento de aplicações, sem a necessidade de mudanças de paradigmas de desenvolvimento.

O Apyon Studio abstrai a complexidade tecnológica permitindo que os engenheiros de *software* trabalhem na maior parte do tempo focados nos requisitos de negócio e não na tecnologia. O Apyon Studio simplifica o processo de construção de aplicações gerando automaticamente mais de 70% do código total da aplicação.

Dentre os benefícios da utilização do Apyon Studio destacam-se:

- Abstração da complexidade tecnológica;
- Simplificação do processo de desenvolvimento de aplicações;
- Redução da dependência tecnológica de fornecedores e profissionais de desenvolvimento;
- Paralelismo no Desenvolvimento - a equipe de negócio pode iniciar a especificação da aplicação enquanto a equipe de tecnologia define os "*patterns*", *lay-outs* dos objetos de tela e os requisitos de desempenho;
- Documentação ativa;
- Aumento da qualidade e padronização do código gerado;
- Menor custo de manutenção e redução nos custos de evolução tecnológica;
- Preservação do investimento feito em outras ferramentas;
- Baixa curva de aprendizado e rápido retorno de investimento;
- Redução dos riscos de fracasso dos projetos.

Do ponto de vista de Pesquisa e Desenvolvimento, o Apyon Studio possui projetos de inovação tecnológica, como o intitulado Modelo Gráfico de Dependência de Regras de Negócio, e o Impacto Físico/Financeiro sobre a Manutenção [STRINGHINI 2003b], que está sendo fomentado pela Fundação de Amparo a Pesquisa do Estado de SP dentro do Programa de Inovação em Pequenas Empresas (PIPE) e o intitulado Definição e implementação de Módulos de Mapeamento Objeto-Relacional no Apyon Studio [STRINGHINI 2003a] é fomentado pelo CNPq dentro do Programa RHAE-Inovação.

O Apyon Studio possui uma equipe focada em Pesquisa e Desenvolvimento para continuar evoluindo o produto, além de ter outros trabalhos sendo iniciados em outras universidades. Para isto o Apyon Studio fornece uma API que pode ser estendida para fornecer novas funcionalidades.

A Apyon Technology, empresa que divulga o Apyon Studio, comercializado no Brasil e está no estágio inicial de expansão internacional iniciando pelo mercado americano, foi citada pelo estudo MIT/Softex [MIT 2003], que analisou os mercados de TI no Brasil, China e Índia como um exemplo brasileiro de inovação tecnológica de classe mundial.

A versão 4.0 do Apyon Studio trata o desenvolvimento de novas aplicações de *software* independente de tecnologia através de seus geradores de código baseado em *templates*. Neste projeto de pesquisa as descrições do modelo de negócio da aplicação legada são importadas para o Apyon Studio para que seja gerada na nova linguagem, inserindo-o no contexto de modernização de aplicações legadas.

Capítulo 3

AMGraA: Uma Abordagem para Migração Gradativa de Aplicações Legadas

Considerando as experiências no projeto de Reengenharia de S*oftware* usando transformações (RST) [FONTANETTE et al. 2001] e baseado nos estudos realizados para esta pesquisa, foi definida a abordagem para a migração gradativa de aplicações legadas comerciais, AMGraA, objetivando atender os requisitos definidos para uma abordagem de migração ideal, apresentados na seção 2.4.

A abordagem AMGraA baseia-se na separação do negócio da aplicação da sua parte tecnológica. Esta separação permite uma migração gradativa das aplicações, reduzindo a complexidade envolvida nos processos de modernização de aplicações legadas e também reduzindo o tempo da migração, uma vez que os resultados são acompanhados gradativamente, conforme a migração da aplicação.

A abordagem compreende 6 fases: Avaliar Aplicação Legada, e Construir Domínio e Transformadores, são fases preparatórias para a realização da migração; Analisar Aplicação Legada, Planejar Projeto de Migração, Migrar Aplicação Legada e Validar Migração da Aplicação são fases da realização da migração, conforme é apresentado no modelo de atividades da Figura 20.

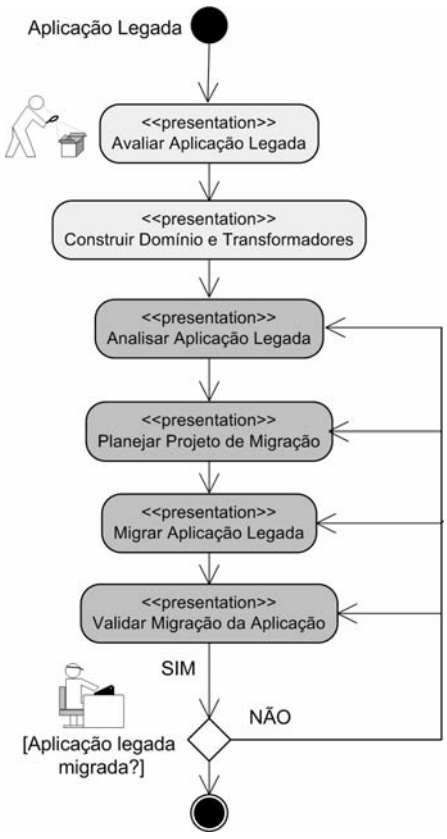


Figura 20 – Diagrama de Atividades da Abordagem AMGraA

O Engenheiro de *Software* é apoiado por três ferramentas, que, integradas, auxiliam na execução da abordagem e compõem um ambiente para a modernização de aplicações legadas. São elas: o sistema de transformação ST Draco-PUC, o Apyon Studio, ambiente de desenvolvimento de aplicações em alto nível de abstração e a ferramenta Migration Project Planning (MPP), construída, neste trabalho, especificamente para complementar o processo, auxiliando na apresentação dos dados do modelo de negócio e no planejamento da migração.

A abordagem baseia-se em técnicas de reengenharia e é apoiada por mecanismos que auxiliam o Engenheiro de *Software* no processo, automatizando grande parte das atividades de modernização.

Na Figura 21 tem-se o modelo de execução da realização da migração, referentes às quatro últimas fases da abordagem proposta. A integração das ferramentas é realizada através de arquivos XML[15] em dois pontos da abordagem, conforme a Figura. Na fase de análise a aplicação legada é submetida ao ST Draco-PUC, que analisa a aplicação produzindo as Descrições XML do Modelo de Negócio da Aplicação Legada. As descrições são importadas na ferramenta Migration Project Planning (MPP), onde o Engenheiro de *Software* vai planejar sua estratégia de migração, definindo as etapas do projeto. O planejamento é armazenado no banco de dados do MPP e as Descrições XML do Planejamento da Estratégia de Migração da etapa a ser migrada é importada no Apyon Studio para a migração da aplicação. Na migração, o Apyon Studio gera a parte referente à tecnologia da aplicação, deixando-a totalmente funcional. O ST Draco-PUC pode ser usado para a conversão das regras de negócio da aplicação. Na validação, o Engenheiro de *Software* define e aplica casos de teste para validar a migração.

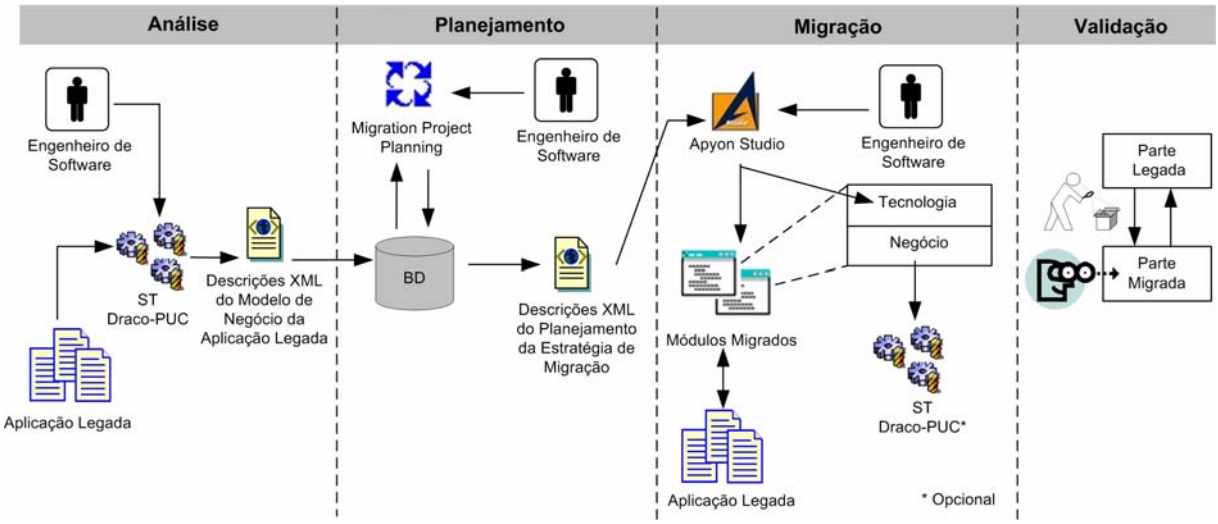


Figura 21 – Modelo de Execução da Abordagem AMGraA

Neste contexto, segue-se uma apresentação das fases que compõem a abordagem proposta.

3.1 Avaliar Aplicação Legada

Esta fase preparatória é essencial para que o Engenheiro de *Software* possa entender o código da aplicação para conhecer suas necessidades e possibilidades relacionadas ao cenário de migração desejado e então apresentar opções e possíveis soluções para o problema.

Nesta fase o Engenheiro de *Software* avalia o código legado da aplicação, procurando respostas para questões, relacionadas:

- **a estrutura do código legado**;
- **a possibilidade de alteração do código legado**. Seria possível alterar a parte inicial do código para que este, por exemplo, pudesse receber parâmetros de uma outra aplicação?
- **ao armazenamento das informações da aplicação legada**. A aplicação persiste suas informações em arquivos ou banco de dados?
- **as formas possíveis para integração**. Considerando que a migração da aplicação será realizada gradativamente, é necessário possibilitar a integração entre migrados e legados, a integração com um banco de dados ou mainframe. Conforme as necessidades da aplicação e requisitos de migração, relacionados ao cenário desejado, o Engenheiro de *Software* pode indicar algumas formas de integração disponíveis, tais como o uso de: *Wrappers*, *Screen-Scrapers* e *Web Services*.

Esta fase é somente uma preparação, onde o Engenheiro de *Software* terá o primeiro contato com a aplicação para conhecê-la, entender seu funcionamento e conhecer as possíveis formas de integração. Portanto, ao longo do processo de migração, conforme as necessidades da aplicação e requisitos de migração, o Engenheiro de *Software* especifica a forma de integração a ser utilizada para criar uma camada de comunicação com a parte legada da aplicação ou ainda com outras aplicações.

- **a inserção de pontos de controle**. Esta atividade é realizada pelo Engenheiro de *Software*, e embora não obrigatória, visa facilitar a construção dos transformadores de código, através da inserção de pontos de controle ao longo do código da aplicação legada.
- **a decomposição em módulos**. A aplicação legada está particionada em módulos para que a abordagem possa ser aplicada? Caso contrário, o Engenheiro de *Software* deverá aplicar o passo Decomposição da aplicação legada em módulos.

[15] Extensible Markup Language (XML) - http://www.w3.org/XML/, Acessado em 15/05/2004.

Ao final desta fase, o Engenheiro de Software tem conhecimento da aplicação legada e sua estrutura. Caso a aplicação esteja particionada em módulos, os artefatos desta fase serão além do conhecimento adquirido pelo Engenheiro de Software, os módulos da aplicação. Se forem necessários os pontos de controle, teremos os códigos dos módulos da aplicação com os pontos de controle inseridos para facilitar a identificação pelos transformadores.

Caso a aplicação legada não esteja particionada em módulos, o Engenheiro de Software aplicará o próximo passo Decomposição da Aplicação Legada em Módulos para particionar a aplicação.

3.1.1 Decomposição da Aplicação Legada em Módulos

Caso a aplicação legada não esteja particionada em módulos, o Engenheiro de *Software* deverá decompor a aplicação em módulos, considerando algum critério como, por exemplo, funcionalidade.

Para decompor a aplicação em módulos devem ser identificadas as dependências entre os módulos. Entretanto, esta é uma tarefa difícil, pois está relacionada ao tamanho da aplicação e a qualidade do código. Casos em que o código possui baixa coesão ou alto acoplamento tornam a tarefa mais difícil, pois a responsabilidade de um módulo encontra-se distribuída pelo código, influenciando o número de dependências. O grande número de dependências impacta na decomposição da aplicação em módulos, tornando a decomposição mais difícil.

Assim, o Engenheiro de *Software* pode decompor a aplicação legada de duas maneiras:

- **Obter a hierarquia de chamadas da aplicação**: o código da aplicação legada é submetido ao ST Draco-PUC para inicialmente identificar a hierarquia de chamadas das unidades de programas que compõem a aplicação. Em seguida, com base nesta hierarquia faz-se a decomposição da aplicação em módulos, ou;
- **Obter informações da equipe que desenvolveu a aplicação**: a equipe que desenvolveu a aplicação conhece as interdependências existentes no código e pode especificar os fontes ou arquivos pertencentes a cada módulo da aplicação.

A decomposição pode ser lógica, baseada na hierarquia de chamadas, ou física, baseada na separação dos programas em diretórios, que representam os módulos da aplicação.

Em casos onde os módulos estão muito integrados, ou seja, inserem dados em arquivos ou tabelas de banco de dados de outros módulos, ou fazem chamadas internas a programas de outros módulos, podem ser necessárias alterações no código da aplicação, ou seja, uma reestruturação no código.

3.2 Construir Domínio e Transformador

Esta fase preparatória é essencial para a preparação do ambiente para a execução do processo. O Engenheiro de *Software* constrói o domínio, através de uma gramática para a linguagem dos programas, no ST Draco-PUC. Documentações e manuais sobre a linguagem do domínio e as regras sintáticas e semânticas da linguagem, orientam a edição da gramática.

Nesta fase o Engenheiro de *Software* edita as regras de transformações para a construção do transformador. O transformador construído é responsável por identificar e coletar descrições sobre o modelo de negócio da aplicação. A partir dessas descrições, será realizado o planejamento do projeto de migração da aplicação na fase de Planejar Projeto de Migração.

Para orientar o Engenheiro de *Software* na identificação e coleta das descrições do modelo de negócio da aplicação, foi definido um modelo para representar os elementos recuperados de uma aplicação legada, conforme é mostrado nas Figuras 22, 23 e 24.

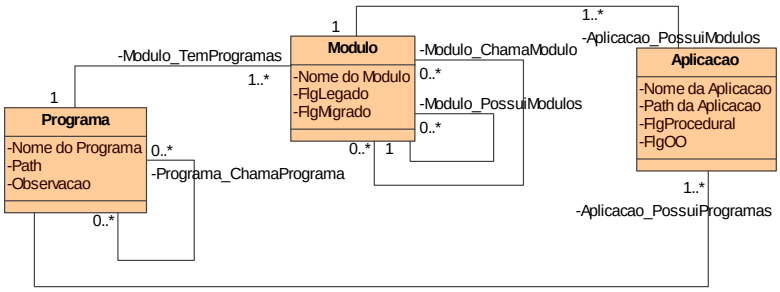


Figura 22 - Descrições do Modelo de Negócio da Aplicação Legada (Parte 1)

Conforme apresentado na Figura 22, segundo o modelo, a Aplicação é composta por Módulos e Programas. Entenda-se por Módulos, subconjuntos da aplicação agrupados por funcionalidades especificas executadas na aplicação. Um Módulo pode solicitar a execução de outros Módulos. Analogamente, Programas podem chamar outros Programas.

O código legado da aplicação possui elementos que podem ser recuperados como: Tabelas, seus Atributos, seus Relacionamentos, as Interfaces da aplicação, os AtributoInterface, os Procedimentos, as RegraDeNegocio, os CasoUso, seus Atores e sua Origem, conforme apresentado na Figura 23.

Os elementos recuperados são extraídos da análise dos Programas da aplicação.

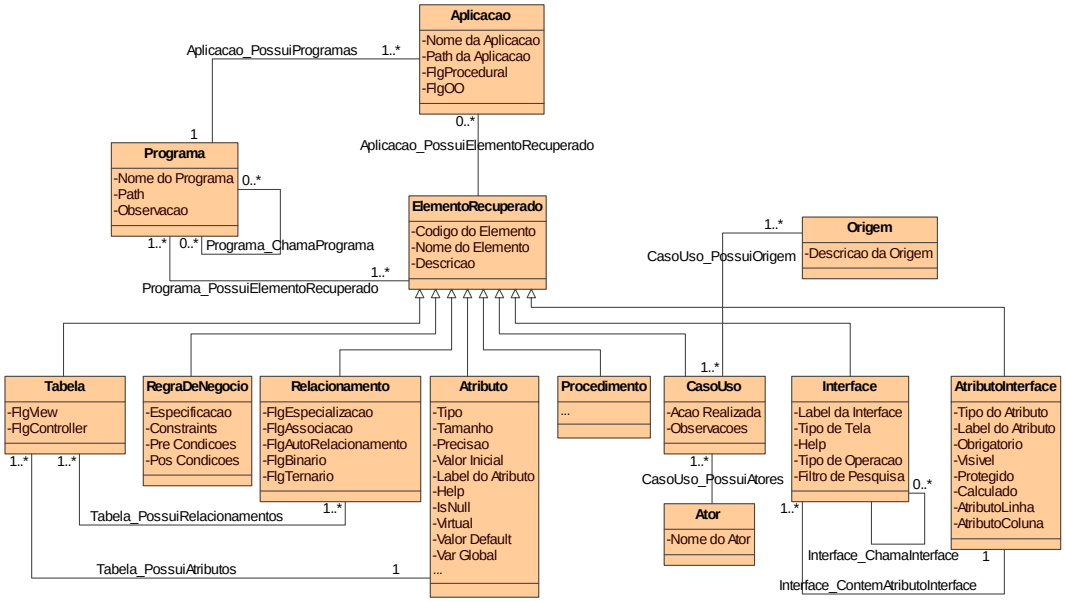


Figura 23 - Descrições do Modelo de Negócio da Aplicação Legada (Parte 2)

Conforme apresentado na Figura 24, as interfaces de usuário da aplicação, representadas aqui pela classe Interfaces possuem atributos de interface, ou seja, AtributoInterface, e podem chamar outras interfaces. Uma interface pode ou não ter Menu e Parâmetros.

O transformador para coleta das descrições do modelo de negócio da aplicação baseia-se no modelo das Figuras 22 a 24 e nas informações da gramática do domínio. Suas transformações identificam e extraem, do código da aplicação legada, dados sobre o modelo de negócio da aplicação.

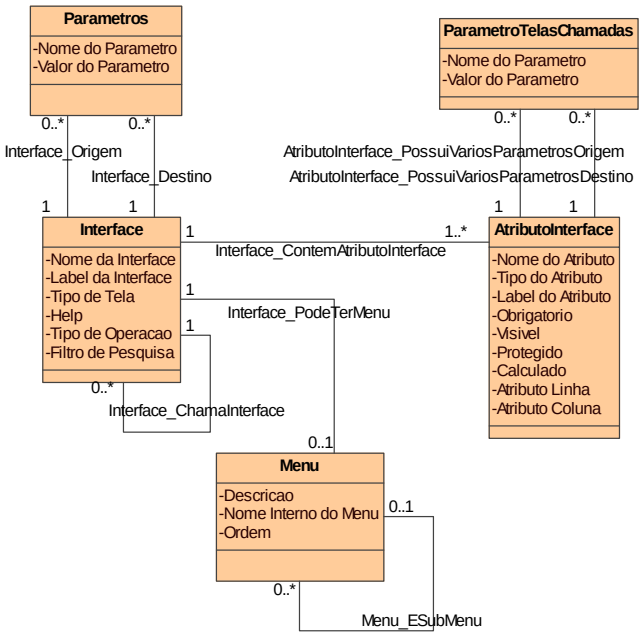


Figura 24 - Descrições do Modelo de Negócio da Aplicação Legada (Parte 3)

Os dados coletados são armazenados como fatos na Base de Conhecimento (KB) do sistema de transformação.

A seguir são apresentadas as atividades que orientam na identificação e coleta das descrições do modelo de negócio da aplicação.

- **Identificar tabelas e campos**: as tabelas e campos podem ser obtidos a partir de:
  - **Arquivos de dados**: as tabelas são identificadas com base nos comandos de abertura e criação de arquivos de dados. Os campos são identificados com base nos campos destes arquivos.
  - **Banco de Dados**: as tabelas são identificadas com base nos comandos de abertura e criação das tabelas de banco de dados. Os campos são identificados com base nos campos destas tabelas.
  - **Estruturas de Dados**: as tabelas são identificadas com base nos comandos de definição de estruturas de dados. Os campos são identificados com base nos campos destas estruturas de dados.
- **Identificar regras de negócio**: as regras de negócio são reconhecidas pela definição de unidades de programa, que pode ser: um programa, subprograma, procedimento, função ou bloco de comandos. Uma unidade de programa pode ser chamada por um evento de interface, por outra unidade de programa, ou pelos *triggers,* que são programas disparados por eventos de criação, alteração, exclusão, pesquisa e replicação do banco de dados.
- **Identificar relacionamentos**: os relacionamentos são reconhecidos pelos campos que formam um índice único em um determinado arquivo ou tabela A, e também são atributos de um outro arquivo ou tabela. Para arquivos ou tabelas que estão relacionados, mas não possuem os mesmos atributos, são também analisados os comandos de acesso aos arquivos ou banco de dados para reconhecer um possível relacionamento.
- **Identificar hierarquia de chamadas dos programas**: a hierarquia de chamadas é definida pela seqüência de execução das unidades de programa do código legado;
- **Identificar fluxo de execução da aplicação legada**: o fluxo de execução de uma aplicação é identificado por comandos que determinam como será a execução da aplicação;
- **Identificar casos de uso**: os casos de uso podem ser obtidos a partir dos:
  - **Cenários da aplicação**: cada opção do menu da aplicação dá origem a um caso de uso.
  - **Scripts de Linkedição**: algumas aplicações legadas possuem scripts de linkedição que definem o relacionamento de vários programas fontes que darão origem a um único programa executável. Cada agrupamento deste dará origem a um caso de uso da aplicação. O

Engenheiro de *Software* com o auxilio do ST Draco-PUC lê os fontes deste script e armazena na base de conhecimento as informações dos casos de uso e os programas fonte que os implementam.

- **Identificação manual**: os casos de uso são determinados manualmente, por meio da hierarquia de chamadas do programas.

- **Identificar interfaces e objetos de interface**: a interface e os objetos de interface de uma determinada linguagem são identificados a partir dos comandos existentes na linguagem para definir a interface e seus objetos. Caso a linguagem não possua tais comandos, a identificação é realizada através de comandos que mostram informações na tela como, por exemplo, o comando *DISPLAY*.

Estas atividades são implementadas no transformador, que atua no código da aplicação legada, identificando e armazenando as informações na KB.

O ST Draco-PUC torna operacional a aplicação das transformações no código da aplicação legada, e, consequentemente, a identificação dos dados do modelo de negócio da aplicação, automatizando, portanto, grande parte das tarefas do Engenheiro de *Software*.

Maiores detalhes sobre a construção dos domínios e transformadores no sistema de transformação podem ser encontrados em [FONTANETTE et al. 2001]. Este trabalho, bem como as experiências anteriores com o ST Draco-PUC [FUKUDA 2000], [JESUS 2000], [NOGUEIRA 2002] e [NOVAIS 2002] serviram de base na definição dos passos para a construção do domínio e do transformador. O capítulo 4 apresenta com detalhes a construção do domínio COBOL e seu transformador construídos para avaliação da abordagem AMGraA.

Caso o ST Draco-PUC já possua o domínio e o transformador da linguagem, o Engenheiro de *Software* pode submeter o código da aplicação legada ao ST Draco-PUC para que a gramática da linguagem, baseada na sintaxe dos comandos dos programas, seja validada. Se forem identificados problemas em alguns dos comandos e estes já estiverem mapeados nas bibliotecas de transformação (transformador), então o Engenheiro de *Software* deverá alterar também as bibliotecas de transformação.

Uma outra alternativa, menos formal, consiste em contratar um especialista no domínio da linguagem para analisar o código fonte e compará-lo com a gramática atual mapeada.

Uma vez realizados os passos preparatórios, o Engenheiro de *Software* dá início a execução do processo de modernização da aplicação.

3.3 Analisar Aplicação Legada

A fase de análise tem como objetivo percorrer o código da aplicação legada recuperando as descrições do modelo de negócio da aplicação legada. Estas descrições consistem de informações que expressam o negócio da empresa, independente da tecnologia e plataforma em que a aplicação foi implementada.

O Engenheiro de Software seleciona os módulos da aplicação que deseja estar migrando primeiro, para então dar início ao processo de migração gradativa. Geralmente, parte-se dos módulos com maior prioridade de migração para a empresa, seguidos dos demais módulos nas próximas iterações.

Assim, nesta fase da abordagem, que consiste no reuso das transformações, considerando os módulos selecionados, parte-se do Código da Aplicação Legada para obter as Descrições XML do Modelo de Negócio da Aplicação Legada, conforme mostrado pela Figura 25. No caso de aplicações não-orientadas a banco de dados, a análise baseia-se nos comandos de acesso aos arquivos utilizados por estas aplicações.

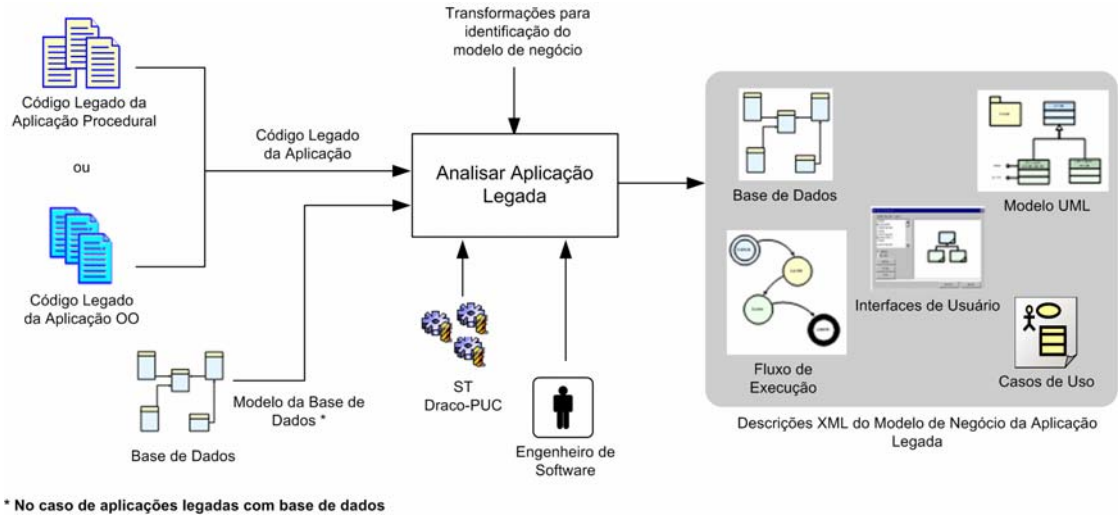


Figura 25 - Analisar Aplicação Legada

O Engenheiro de *Software* é apoiado pelo ST Draco-PUC na análise do código da aplicação, aplicando transformações do domínio do código legado e coletando os dados do modelo de negócio da aplicação. Estes dados são armazenados como fatos na Base de Conhecimento (KB) do ST Draco-PUC.

Dentre os dados coletados têm-se: descrições sobre a Base de Dados com informações sobre tabelas, campos e relacionamentos, Fluxo de Execução, Interfaces de Usuário, Casos de Uso,

Regras de Negócio, e no caso de aplicações Orientadas a Objetos, têm-se informações sobre o Modelo O.O.

Na Figura 26 é apresentado o modelo de atividades a serem realizadas para a identificação das descrições do modelo de negócio da aplicação.

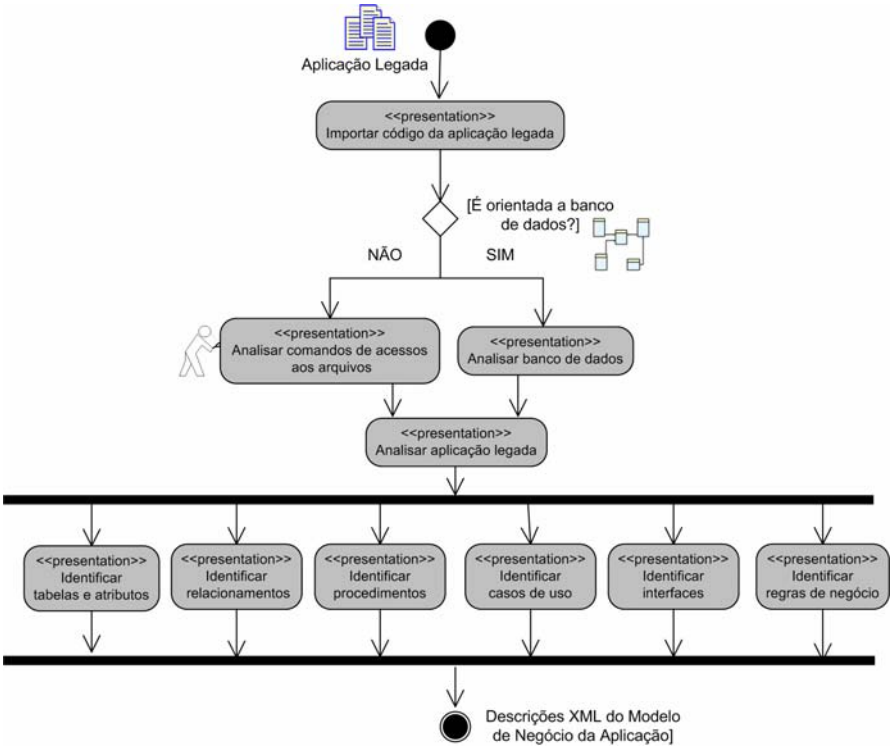


Figura 26 – Modelo de Atividades da fase Analisar Aplicação Legada

Os fatos armazenados na Base de Conhecimento (KB) são transformados em descrições no formato XML no final da análise do código da aplicação. Este arquivo XML segue uma DTD *(Document Type Definition),* conforme apresentada na Figura 27, definida para a importação das descrições do modelo de negócio na ferramenta Migration Project Planning (MPP).

A *tag* ELEMENT define *tags* para o arquivo XML que será validado pela DTD como, por exemplo, a *tag* ElementoRecuperado que representa os elementos recuperados do modelo de negócio da aplicação.

A *tag* ATTLIST define atributos para uma determinada *tag* do arquivo XML como, por exemplo, os atributos codigoElemento, codigoSistema e codigoProjeto da *tag* ElementoRecuperado.

Os atributos de uma *tag* possuem tipos. O tipo PCDATA representa uma seqüência qualquer de letras e números e o tipo IDREF representa um identificador que não pode ser repetido em todo o arquivo XML e é usado para fazer referência. Os atributos possuem ainda valores como: REQUIRED e IMPLIED que significam que um atributo é necessário ou que não precisam ser incluídos no arquivo XML, respectivamente.

O arquivo XML será utilizado pela ferramenta Migration Project Planning (MPP), na próxima fase da abordagem.

```
<!ELEMENT MPPImportacao (ElementoRecuperado)+>
<!ELEMENT ElementoRecuperado (Tabela, Atributo, Interface, AtributoInterface,
Procedimento, RegraDeNegocio, CasoUso, Ator, Parametros, ParametroTelaChamada)+>
<!ATTLIST ElementoRecuperado codigoElemento (#PCDATA) #REQUIRED>
<!ATTLIST ElementoRecuperado codigoSistema (#PCDATA) #REQUIRED>
<!ATTLIST ElementoRecuperado codigoProjeto (#PCDATA) #REQUIRED>

<!ELEMENT Tabela (Atributo)+>
<!ATTLIST Tabela codigoElemento (IDREF) #REQUIRED>
<!ATTLIST Tabela codigoSistema (IDREF) #REQUIRED>
<!ATTLIST Tabela codigoProjeto (IDREF) #REQUIRED>
<!ATTLIST Tabela nomeTabela (#PCDATA)>
<!ATTLIST Tabela descricaoTabela (#PCDATA)>
<!ATTLIST Tabela flgView (S|N) #IMPLIED>
<!ATTLIST Tabela flgController (S|N) #IMPLIED>

<!ELEMENT Atributo>
<!ATTLIST Atributo codigoElemento (IDREF) #REQUIRED>
<!ATTLIST Atributo codigoSistema (IDREF) #REQUIRED>
<!ATTLIST Atributo codigoProjeto (IDREF) #REQUIRED>
<!ATTLIST Atributo nomeAtributo (#PCDATA)>
<!ATTLIST Atributo descricaoAtributo (#PCDATA)>
<!ATTLIST Atributo tipoAtributo (#PCDATA)>
<!ATTLIST Atributo tamanhoAtributo (#PCDATA)>
<!ATTLIST Atributo precisao (#PCDATA) #IMPLIED>
<!ATTLIST Atributo valorInicial (#PCDATA) #IMPLIED>
<!ATTLIST Atributo labelAtributo (#PCDATA)>
<!ATTLIST Atributo help (#PCDATA) #IMPLIED>
<!ATTLIST Atributo isNull (S|N) #IMPLIED>
<!ATTLIST Atributo virtual (S|N) #IMPLIED>
<!ATTLIST Atributo valorDefault (#PCDATA) #IMPLIED>
<!ATTLIST Atributo varGlobal (#PCDATA) #IMPLIED>
<!ATTLIST Atributo codigoElementoTabela (#PCDATA)>

<!ELEMENT Interface (AtributoInterface)+>
<!ATTLIST Interface codigoElemento (IDREF) #REQUIRED>
<!ATTLIST Interface codigoSistema (IDREF) #REQUIRED>
<!ATTLIST Interface codigoProjeto (IDREF) #REQUIRED>
<!ELEMENT Interface nomeInterface (#PCDATA)>
<!ELEMENT Interface labelInterface (#PCDATA)>
...
```

Figura 27 - DTD definida para importação dos dados do modelo de negócio da aplicação legada no MPP

3.4 Planejar Projeto de Migração

Esta fase consiste em planejar o projeto de migração da aplicação legada. Para auxiliar o Engenheiro de *Software* foi construída a ferramenta Migration Project Planning (MPP) apresentada a seguir.

3.4.1 Migration Project Planning (MPP)

A ferramenta Migration Project Planning (MPP) foi construída ao longo deste projeto de pesquisa para apoiar o Engenheiro de *Software* no planejamento de seus projetos de migração.

O MPP foi desenvolvido no Apyon Studio, utilizando a linguagem C# na plataforma .NET e o SGBD SQL Server na versão 2000.

Na Tabela 3 são apresentados os principais casos de uso definidos para o MPP. Os casos de uso, conforme mostrado na tabela, serão executados pelo principal ator da aplicação, o Engenheiro de *Software*.

Tabela 3 – Principais Casos de Uso do MPP

Nr.	Descrição	Caso de Uso	Eventos	Respostas
01	CadastrarProjeto	Engenheiro de Software cadastra Projeto de Migração de uma aplicação.	dadosProjeto	MSG01
02	CadastrarCamadas	O Engenheiro de Software cadastra as camadas origem e destino das aplicações	dadosCamada	MSG02
03	CadastrarTecnologias	O Engenheiro de Software cadastra tecnologias	dadosTecnologia	MSG03
04	DetalharProjeto	Engenheiro de Software detalha projeto	nomeProjeto, dadosDetalhamento	MSG04
05	ImportarModeloNegocio	O Engenheiro de Software importa as informações do modelo de negócio coletadas	dadosModeloNegocio	MSG05
06	DefinirEtapasDeMigracao	O Engenheiro de Software define as etapas do seu projeto de migração	dadosEtapa	MSG06
07	ExportarDescricoesEtapa	O Engenheiro de Software exporta as descrições da etapa que pretende migrar	dadosExportacao	MSG07
08	CadastrarModulos	O Engenheiro de Software cadastra os módulos da aplicação	dadosModulo dadosProgramas	MSG08
09	AnalisarDadosInterfaces	O Engenheiro de Software analisa dados coletados para interfaces de usuário da aplicação	codigoInterface	dadosInterface dadosAtributos
10	AnalisarDadosTabelas	O Engenheiro de Software analisa dados coletados para as tabelas	codigoTabela	dadosTabela dadosCampos

MSG01: Projeto cadastrado com sucesso / Projeto atualizado com sucesso / Projeto excluído com sucesso.

MSG02: Camada cadastrada com sucesso / Camada atualizada com sucesso / Camada excluída com sucesso.

MSG03: Tecnologia cadastrada com sucesso / Tecnologia atualizada com sucesso / Tecnologia excluída com sucesso

MSG04: Detalhamento cadastrado com sucesso / Detalhamento atualizado com sucesso / Detalhamento excluído com sucesso.

MSG05: Dados importados com sucesso / Dados atualizados com sucesso / Dados excluídos com sucesso

MSG06: Dados exportados com sucesso / Dados não exportados

MSG07: Etapa cadastrada com sucesso / Etapa atualizada com sucesso / Etapa excluída com sucesso.

MSG08: Módulo cadastrado com sucesso / Módulo atualizado com sucesso / Módulo excluído com sucesso.

MSG09: Interface cadastrada com sucesso / Interface atualizada com sucesso / Interface excluída com sucesso.

MSG10: Tabela cadastrada com sucesso / Tabela atualizada com sucesso / Tabela excluída com sucesso.

Na Figura 28 tem-se a interface principal da ferramenta MPP. O menu de opções está dividido em: Menu de Controle, Menu de Planejamento e Menu para Visualização das descrições do modelo de negócio da aplicação, obtidas pelo ST Draco-PUC. À direita tem-se a Área de Trabalho da Aplicação.

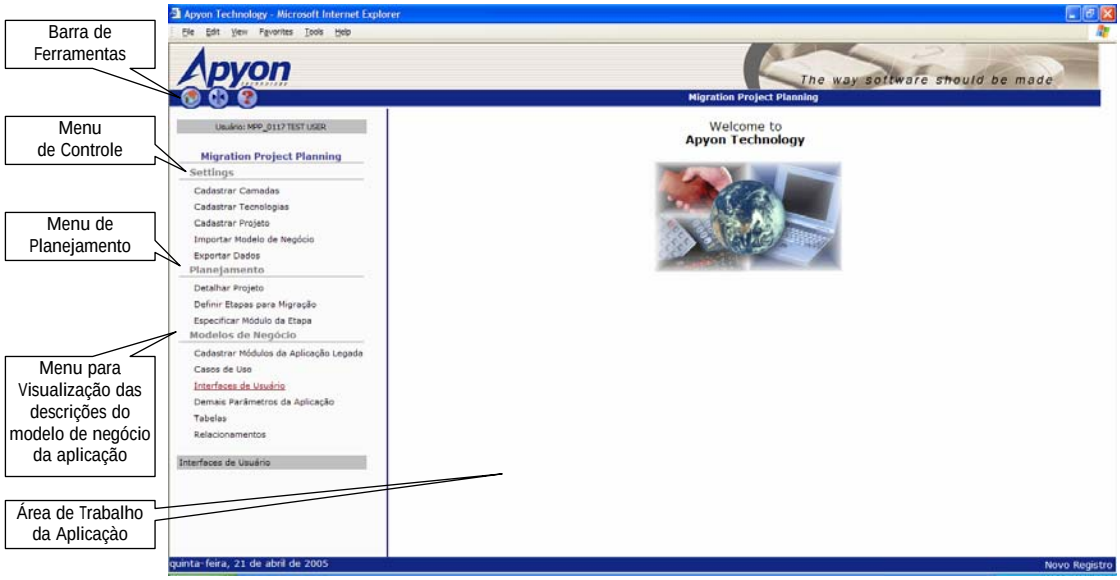


Figura 28 – Interface principal do MPP

Os demais artefatos elaborados para a construção da ferramenta MPP encontram-se no Apêndice.

3.4.2 Planejar Projeto de Migração

Na Figura 29 tem-se a fase de Planejar Projeto de Migração. Esta fase é dividida nas seguintes atividades, conforme apresentado no modelo de atividades da Figura 30:

- **Cadastrar projeto de migração**: o Engenheiro de *Software* cadastra um projeto de migração para a aplicação legada, informando dados quanto à sua estrutura, como o número de camadas em que a aplicação foi desenvolvida e as tecnologias utilizadas no seu desenvolvimento, seus *paths*,, detalhando o projeto da aplicação legada;

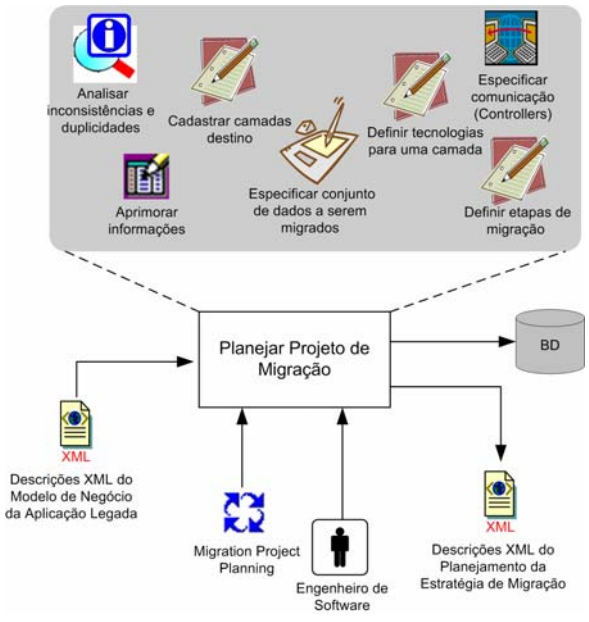


Figura 29 - Planejar Projeto de Migração

- **Importar descrições do modelo de negócio da aplicação legada**: o Engenheiro de *Software* importa as descrições XML do modelo de negócio da aplicação legada. Essas descrições podem conter informações de um ou mais módulos;
- **Analisar informações coletadas**: o Engenheiro de *Software* analisa as informações coletadas na fase anterior e pode então alterar os nomes para nomes mais significativos, corrigir inconsistências e duplicidades que possam existir devido às várias manutenções sobrepostas em aplicações legadas;
- **Aprimorar informações para atender a novas necessidades de negócio**: o Engenheiro de *Software* pode re-especificar o modelo de negócio da aplicação para atender às novas necessidades de negócio como, por exemplo, para alterar o fluxo de execução da aplicação, inserir uma nova classe ou componente, e inserir novas funcionalidades para a aplicação;
- **Definir camadas**: o Engenheiro de *Software* após analisar a aplicação legada informa ao MPP as camadas que compõem a arquitetura da aplicação legada, definindo e fornecendo à ferramenta MPP as camadas para a arquitetura da nova aplicação;
- **Definir tecnologias para cada camada**: o Engenheiro de *Software,* após analisar a aplicação legada, informa ao MPP as tecnologias utilizadas no desenvolvimento da aplicação legada, definindo e fornecendo à ferramenta MPP as tecnologias a serem utilizadas na nova aplicação;

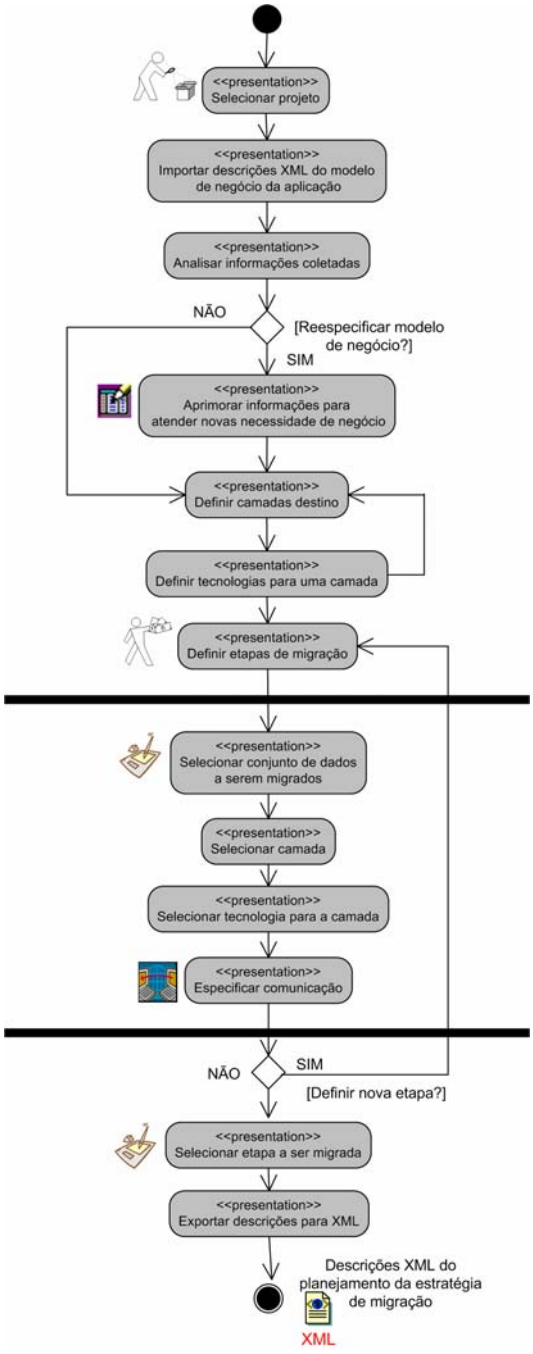


Figura 30 - Modelo de atividades da fase Planejar Projeto de Migração

- **Definir etapas de migração**: o Engenheiro de *Software* define as etapas do seu projeto de migração e então define os dados a serem migrados em cada etapa;
- **Selecionar conjunto de dados a serem migrados**: o Engenheiro de *Software* define o conjunto de dados a ser migrado, indicando o módulo e os programas deste módulo.

- **Selecionar camada para a migração**: o Engenheiro de *Software* especifica as camadas para o projeto de migração;
- **Selecionar tecnologia para a camada**: o Engenheiro de *Software* especifica as tecnologias a serem utilizadas no desenvolvimento da nova aplicação;
- **Especificar comunicação**: o Engenheiro de *Software* especifica a forma de comunicação entre o módulo migrado e a aplicação legada e o banco de dados como, por exemplo, um *controller*, ou um novo componente. Existem muitas formas para realizar tal comunicação atualmente, principalmente protocolos para compartilhar informações entre ambientes heterogêneos. Isto pode ser feito através de: uma "emulação de telas", um novo componente para o mainframe, *Web Services*, ou arquivos texto de comunicação. Portanto, novos componentes podem ser criados para atender às necessidades da migração gradativa, dependendo do cenário exigido pelo projeto de migração;
- **Selecionar etapa a ser migrada**: o Engenheiro de *Software* define a etapa a ser migrada. Esta atividade depende de fatores externos ao processo, pois está voltado às necessidades e prioridades da empresa. Portanto, o Engenheiro de *Software* define métricas e prioriza as etapas de migração, baseado nas métricas para o cenário do projeto de migração. Por exemplo, se o objetivo da migração é disponibilizar parte da aplicação na *Web*, então o Engenheiro de *Software* poderá considerar as necessidades do cliente para priorizar as etapas de migração;
- **Exportar descrições do planejamento da estratégia de migração**: o Engenheiro de *Software* exporta os dados do planejamento da etapa a ser migrada para descrições XML. Estas descrições contêm informações sobre os elementos recuperados para os programas relacionados ao módulo definido para esta etapa;

Em resumo, esta fase inicia-se quando o Engenheiro de *Software* importa as Descrições XML do Modelo de Negócio da Aplicação Legada para o projeto definido. Após a importação, o Engenheiro de *Software* pode então analisar os dados recuperados, alterar os nomes para nomes mais significativos, e corrigir inconsistências. O Engenheiro de *Software* pode ainda alterar os nomes de campos para nomes mais significativos, e ainda, re-especificar o modelo de negócio da aplicação, adicionando novas funcionalidades para atender a novas necessidades de negócio, caso necessário.

Em seguida, o Engenheiro de *Software* define as camadas da nova aplicação e as tecnologias envolvidas do desenvolvimento. O Engenheiro de Software dá inicio ao planejamento de sua estratégia de migração, definindo as etapas do projeto, especificando os dados a serem migrados

em cada etapa do processo de migração e especificando as formas de comunicação dos módulos migrados com a aplicação legada e com outras aplicações.

As especificações realizadas nesta fase são persistidas na base de dados do MPP. Como as especificações da aplicação estão armazenadas, o Engenheiro de *Software* pode alterar as informações quando achar necessário, facilitando manutenções futuras.

Definidas as etapas, o Engenheiro de *Software* seleciona a etapa a ser migrada, exportando seus dados para obter as descrições XML do planejamento da estratégia de migração, que serão importadas no Apyon Studio.

O propósito é que esta ferramenta ofereça flexibilidade para o planejamento da estratégia de migração e auxilie o Engenheiro de *Software* na definição da estratégia de migração, definindo "o que" migrar, "quando" migrar e "como" migrar.

3.5 Migrar Aplicação Legada

Esta fase visa realizar a migração da aplicação, de acordo com as etapas definidas no planejamento. Conforme é apresentado na Figura 31, o Engenheiro de *Software* tem o auxílio das ferramentas Apyon Studio, para a migração de requisitos não-funcionais da aplicação, e do ST Draco-PUC para migração dos requisitos funcionais da aplicação.

As ferramentas Apyon Studio e ST Draco-PUC se complementam, pois o Apyon Studio gera o código de toda a parte tecnológica da aplicação em uma linguagem específica, enquanto que o ST Draco-PUC pode ser utilizado nesta fase para a conversão das regras de negócio da aplicação.

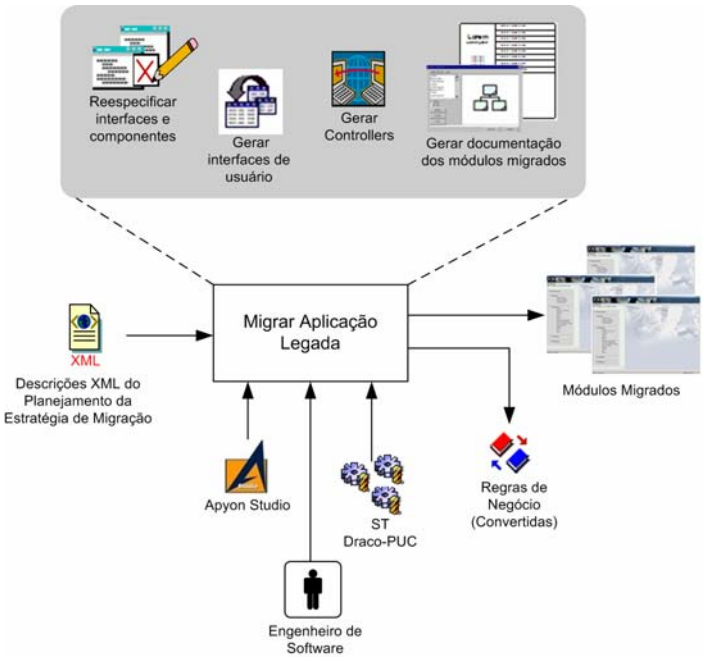


Figura 31 – Migrar Aplicação Legada

Na Figura 31 é apresentada a fase Migrar Aplicação Legada. Esta fase possui as seguintes atividades, conforme mostra o modelo de atividades da Figura 32:

- **Importar as descrições XML do planejamento da estratégia de migração**: o Engenheiro de *Software* importa as descrições XML para o repositório do Apyon Studio;

- **Re-especificar interfaces e componentes**: o Engenheiro de *Software* pode re-especificar as interfaces e componentes da aplicação legada para melhorar sua apresentação e usabilidade, possibilitando através desta atividade aumentar a satisfação do usuário;
- **Especificar novas interfaces e componentes**: o Engenheiro de *Software* pode definir novas interfaces e componentes para atender a novas funcionalidades da aplicação;
- **Gerar interfaces de usuário**: o Engenheiro de *Software* gera as interfaces de usuário na tecnologia definida anteriormente;
- **Gerar componentes**: o Engenheiro de *Software* gera os componentes na tecnologia definida anteriormente;
- **Gerar *controllers***: O Engenheiro de *Software* gera os *controllers* necessários para realizar a comunicação dos módulos migrados;
- **Gerar documentação**: o Engenheiro de *Software* gera a documentação dos módulos migrados e das informações importadas no Apyon Studio;
- **Enviar informações ao ST Draco-PUC**: no caso de conversão das regras de negócio, o Engenheiro de Software envia as especificações das regras de negócio ao ST Draco-PUC;
- **Converter regras de negócio**: o Engenheiro de *Software* pode converter as regras de negócio da aplicação. As regras podem ser migradas aos poucos e independentemente da migração da parte tecnológica da aplicação, considerando o objetivo da migração;
- **Importar regras de negócio para o Apyon Studio**: o Engenheiro de *Software* importa as regras de negócio para o repositório do Apyon Studio, integrando-as ao código da nova aplicação;

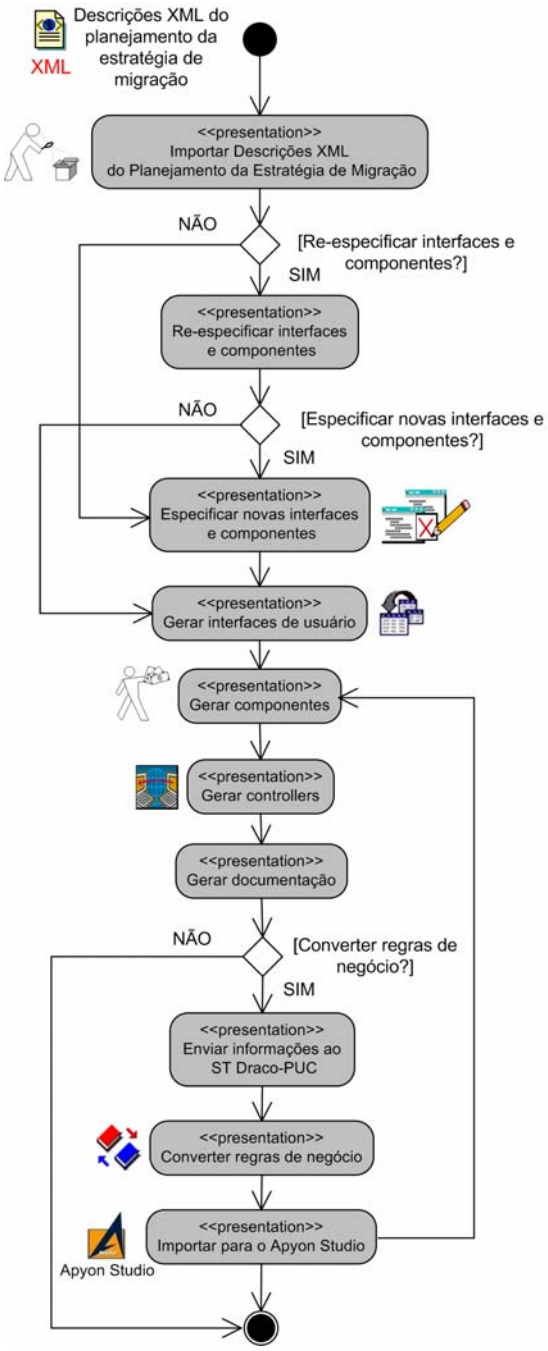


Figura 32 - Modelo de atividades da fase Migrar Aplicação Legada

Em resumo, esta fase inicia-se quando as Descrições XML do Planejamento da Estratégia de Migração, realizadas pelo Engenheiro de *Software* na ferramenta MPP, são importadas para o repositório do Apyon Studio.

Em seguida, o Engenheiro de *Software* pode re-especificar a aplicação no Apyon Studio antes de gerar o seu código, por exemplo, especificando novas interfaces e novos componentes que atendam a novos requisitos. O Apyon Studio gera as interfaces de usuário, os componentes de persistência e os *controllers* necessários para a integração e comunicação com os demais módulos e com as regras de negócio legadas da aplicação. Todas essas informações são adicionadas na documentação gerada pelo Apyon Studio.

A geração de código no Apyon Studio é realizada através de geradores de tecnologia que trabalham com *templates* do tipo *search/replace*, conforme apresentado na Figura 33. Com base nos *templates* são substituídos os elementos do repositório do Apyon Studio. Os geradores são flexíveis e qualquer tipo de arquitetura pode ser gerada, bastando apenas alterações nos *templates*, que são arquivos texto.

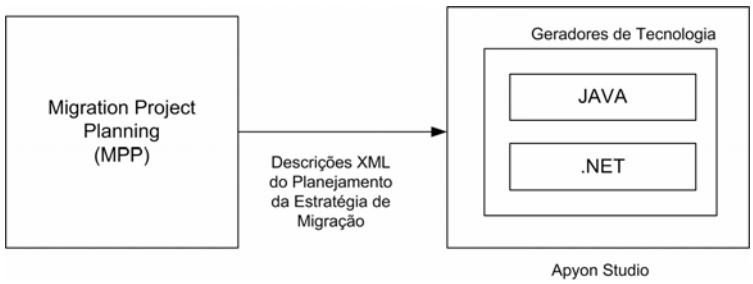


Figura 33 – Geradores de código do Apyon Studio

O Engenheiro de *Software* pode ainda realizar a migração das regras de negócio da aplicação. Esta atividade pode ser opcional, uma vez que o processo é gradativo e a migração das regras de negócio pode ser adiada, de acordo com o cenário da migração. O Engenheiro de *Software* pode migrar as regras de negócio com o auxílio do ST Draco-PUC ou pode reescrevê-las.

É importante considerar que o código gerado para as regras de negócio pelo ST Draco-PUC deve ser analisado e trabalhado, para que a estrutura do código reflita a nova linguagem escolhida e não a linguagem da aplicação legada. Isto é importante para facilitar manutenções futuras no novo código da aplicação.

Esta fase oferece recursos que tornam mais flexível a migração da aplicação, uma vez que a aplicação vai ser migrada de forma gradativa e os requisitos definidos para um projeto de migração podem não ser iguais aos requisitos especificados para outros projetos. Desta forma, pode-se migrar somente requisitos não-funcionais da aplicação, como as interfaces, mantendo os

requisitos funcionais, ou seja, as regras de negócio da aplicação. Assim, é preciso definir como será o acesso às informações legadas através das novas interfaces.

3.6 Validar Migração da Aplicação

Esta fase visa testar e validar a nova aplicação gerada. Após a fase de migração de cada etapa, o Engenheiro de *Software* define e aplica casos de testes para testar e validar a nova parte migrada e implantar o novo módulo migrado da aplicação.

Os itens e casos de testes são elaborados de acordo com o cenário de migração abordado para validar, entre outros quesitos, a integração e a comunicação da parte migrada com a aplicação legada.

O Engenheiro de *Software* deve aplicar tanto os testes caixa-preta, que visam encontrar erros em funções incorretas, erros de interfaces e de comportamento, quanto os testes caixa-branca, que visam encontrar erros na estrutura de controle da aplicação. O Engenheiro de Software deverá fazer uso de outros métodos como particionamento de equivalência, análise do valor limite e testes de regressão.

Para os cenários apresentados na seção 2.6 citam-se algumas diretivas para possíveis casos de testes:

- **Cenário 1**: para este cenário onde somente as interfaces de usuário dos módulos são migradas para *Web*, mantendo inalteradas as regras de negócio e a base de dados legada, pode-se considerar as seguintes diretivas para casos de teste:
  - **Cadastrar instância**: inserir através da interface uma determinada instância e analisar se a resposta emitida pela interface é válida;
  - **Consultar instância**: verificar se para uma determinada informação "x" informada para a interface, tem-se a resposta "y";

- **Cenário 2**: para este cenário onde tanto as interfaces de usuário quanto as regras de negócio são migrados, mantendo o acesso à base de dados legada, pode-se considerar a seguinte diretiva para casos de teste:
  - **Verificar resultado da execução da regra de negócio**: para cada regra de negócio criar casos de testes

- **Cenário 3**: para este cenário onde a base de dados legada, baseada em arquivos, é migrada para um banco de dados, pode-se considerar as seguintes diretivas para casos de testes:

- **Inserir dados**: inserir um determinado dado "x" no banco de dados e analisar se a resposta "y" é válida;
- **Consultar dados**: consultar um determinado dado "x", inserido anteriormente, e analisar se ele é correto;

- **Cenário 4**: para este cenário onde é criado um *screen-scraper* para aumentar a satisfação do usuário quanto à aparência das interfaces caractere, pode-se considerar a seguinte diretiva para casos de testes:
  - **Cadastrar instância**: inserir através da nova interface uma determinada instância e analisar se os campos da interface emulada são os mesmos presentes na interface legada;
  - **Consultar instância**: consultar um determinado dado e verificar se as informações da emulação da interface são as mesmas presentes na interface legada;

O Engenheiro de *Software* documenta os resultados dos testes realizados em um relatório de testes. Este relatório servirá de base para que as falhas encontradas durante o processo de teste possam ser corrigidas.

O capitulo 4 apresenta o estudo de caso realizado em uma aplicação de gestão empresarial escrita na linguagem COBOL, com objetivo de modernizar a aplicação para a plataforma .NET, para validação da abordagem AMGraA.

Capítulo 4

Validação da Abordagem

O domínio escolhido para a validação da abordagem proposta foi o domínio COBOL, uma vez que o número de aplicações escritas nesta linguagem é bem grande (cerca de 30% no Brasil) e as manutenções em código COBOL são custosas.

Dentre as aplicações do domínio COBOL, adotou-se uma aplicação legada de gestão empresarial para indústrias, chamada Ômega, para testar a abordagem.

A aplicação Ômega tem 900 programas fontes, totalizando em torno de 315.000 linhas de código desenvolvidas em COBOL Microfocus 4.5 [CARVALHO 1993], com 280 arquivos de dados (VSAM), conforme é ilustrado na Figura 34.

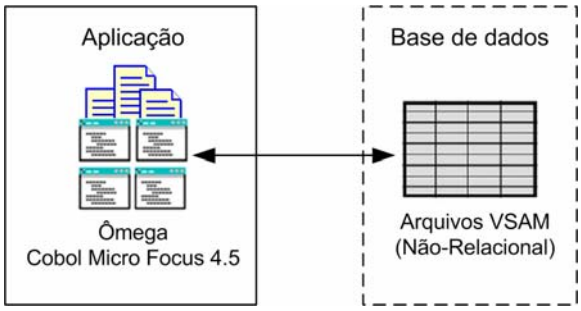


Figura 34 - Aplicação Legada Ômega

A aplicação Ômega possui os seguintes módulos:

- **Básico**, que contém a estrutura básica da aplicação e as informações comuns aos demais módulos;
- **Financeiro**, constituído de: Contas a Pagar, Contas a Receber, Fluxo de Caixa e Conta Corrente de Caixa e Bancos;
- **Materiais**, constituído de: Compras e Estoque;
- **Vendas**, constituído de: Pedidos, Faturamento, Cotas e Conta Corrente de Representantes;
- **Produção**, constituído de: PCP e Controle de Chão de Fábrica;
- **Folha de Pagamento**, constituído de: Folha de Pagamento, Ponto Eletrônico e Benefícios;
- **Contabilidade**, constituído de: Controle de Patrimônio, Contabilidade e Livros Fiscais.

A empresa Orion-ASP [14], proprietária da Ômega, solicitou sua reconstrução para ser executada em plataformas atuais de hardware e *software*. Esta solicitação foi devido aos problemas de manutenções na linguagem COBOL, que estavam cada vez mais custosas, pois os profissionais com experiência nesta linguagem estão migrando para novas linguagens. Assim, dentre os fatores motivadores para a modernização da Ômega destacam-se o uso de uma linguagem de programação moderna e a exploração dos serviços e recursos da Internet. Desta forma, a empresa optou por migrar sua aplicação para a plataforma .NET utilizando a linguagem C#.

Para realizar a migração da aplicação houve a colaboração de 4 participantes com os seguintes perfis:

- Participante 1: profissional com experiência em manutenções da aplicação legada;
- Participante 2: profissional com experiência em desenvolvimento de aplicações na linguagem COBOL;
- Participante 3: profissional com experiência em desenvolvimento de aplicações na plataforma .NET;
- Participante 4: Engenheiro de *Software*.

A aplicação Ômega não possuía documentação e, portanto, foram utilizados na realização da migração os programas fonte e o conhecimento e experiência do Participante 1 na aplicação legada.

O desenvolvimento da migração ocorreu nas dependências do GOES[16] com reuniões esporádicas com os Participantes 1, 2 e 3 em seus locais de trabalho e teve duração de 2 meses e 15 dias.

Segue-se uma apresentação da aplicação da abordagem para a reconstrução da aplicação Ômega.

4.1 Avaliar Aplicação Legada

Primeiramente, o Engenheiro de *Software,* com o auxílio do Participante 1, avaliou a aplicação legada com o propósito de conhecer a aplicação, sua estrutura e funcionalidades.

Nesta fase o Engenheiro de *Software* constatou que a aplicação Ômega estava particionada em módulos e que o código da aplicação era suscetível a alterações e, por isso, seria possível

[16] Grupo de Engenharia de Software – Departamento de Computação – Universidade Federal de São Carlos (UFSCar)

modificá-lo para que a aplicação legada pudesse ser chamada pela nova aplicação a ser gerada, através de um *wrapper* ou ainda um *Web Services*.

4.2 Construir Domínio e Transformador

Nesta fase o Engenheiro de *Software* construiu o domínio COBOL, através da definição da gramática da linguagem baseado em documentações e manuais da linguagem. Em seguida, o Engenheiro de *Software* construiu também o transformador COBOL que irá identificar os dados do modelo de negócio da aplicação.

Segue-se uma breve apresentação da linguagem COBOL.

4.2.1 Linguagem COBOL (Common Business Oriented Language)

A linguagem COBOL *(Commom Business Oriented Language)* foi criada em 1959 para ser aplicada em problemas comerciais. COBOL é uma linguagem de 3ª geração e faz uso de comandos com nomes auto-explicativos como: WRITE, READ, OPEN e CLOSE. É uma linguagem compilada, exigindo, portanto, um compilador para ser executada.

Os programas escritos em COBOL podem ter 4 divisões, conforme apresentado na Figura 35.

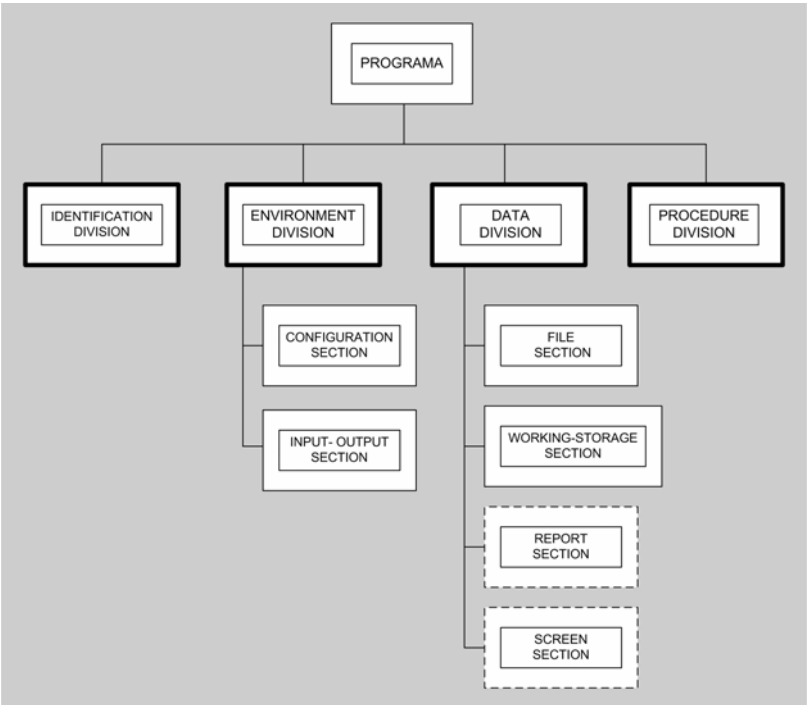


Figura 35 – Estrutura das divisões de um programa COBOL

Cada divisão possui funções específicas. Estas divisões podem ainda ser divididas em seções (SECTIONS) e estas em parágrafos. Todas as outras instruções do programa são consideradas declarações COBOL.

Os nomes de divisões, seções e parágrafos devem ser codificados na margem A (coluna 8). Todas as outras declarações são codificadas na margem B (coluna 12).

Segue uma breve apresentação das divisões e seções. Vale ressaltar que as divisões devem aparecer nesta ordem dentro de um programa

1. IDENTIFICATION DIVISION: Esta divisão identifica o programa-fonte, com dados sobre o autor, data em que foi escrito, observações sobre o que o programa faz, e sobre a segurança. Na Figura 36 é apresentado um trecho desta divisão.

```
7   8       12
    IDENTIFICATION DIVISION.
    PROGRAM-ID. nome do programa.
    [AUTHOR. nome do programador.]
    [INSTALLATION. nome da empresa ou local de geração do programa.]
    [DATE-WRITTEN. data em que o programa foi escrito.]
    [DATE-COMPILED. data em que o programa foi compilado.]
    [SECURITY. comentários sobre a segurança do programa e/ou seus arquivos.]
    [REMARKS. comentários adicionais sobre o programa.]
```

Figura 36 – Identification Division

2. ENVIRONMENT DIVISION: Esta divisão fornece informações relativas aos meios externos, ou seja, arquivos e equipamento. Define os arquivos a serem utilizados no programa, a sua organização, meio de acesso, chaves primárias e/ou secundárias. Na Figura 37 é apresentado um trecho desta divisão.

```
7   8       12
    ENVIRONMENT DIVISION.
    [CONFIGURATION SECTION.]
    [SOURCE-COMPUTER. computador a ser utilizado na compilação do programa.]
    [OBJECT-COMPUTER. computador onde o programa será executado.]
    [SPECIAL NAMES
            [CURRENCY SIGN IS literal]                                      Obs 1
            [DECIMAL-POINT IS COMMA.]                                       Obs 2
            [PRINTER  IS  nome  externo  (mnemônico)  para  referenciação  da
            impressora.] ]
    [INPUT-OUTPUT SECTION.]
    [FILE-CONTROL
            [SELECT nome do arquivo ASSIGN TO {DISK}
                                                    {PRINTER} ]       Obs 3
            [ORGANIZATION IS {SEQUENTIAL}
                             {INDEXED}
                             {RELATIVE} ]                             Obs 4
            [ACCESS MODE IS {SEQUENTIAL}
                             {RANDOM}
                             {DINAMIC} ]                              Obs 5
            [RECORD KEY IS nome de um campo ou conjunto de campos índice do arquivo]
            [FILE STATUS IS nome de uma variável para armazenamento do status do arquivo.] ]
```

Figura 37 – Environment Division

Obs1: substitui na cláusula PICture (DATA DIVISION) o sinal corrente "$" pela literal especificada (por exemplo: "R$").

Obs2: substitui o ponto decimal utilizado na notação americana, pela vírgula utilizada na notação nacional;

Obs3: DISK: se o arquivo de leitura ou gravação for direcionado para disco; PRINTER: se for um arquivo a ser impresso.

Obs4: SEQUENTIAL: os registros só podem ser pesquisados um após o outro; INDEXED/RANDOM: os registros são pesquisados sem uma ordem aparente, através de um arquivo de índices relacionado ao arquivo principal;

Obs5: RELATIVE/DINAMIC: idem.

3. **DATA DIVISION**: Esta divisão armazena todos os dados a serem processados ou manipulados pelo programa, durante o processamento, podendo ser internos ou externos. A organização do arquivo é estabelecida no momento em que o arquivo é criado, e não pode ser modificada. Quando a organização não é especificada, a organização seqüencial é assumida na compilação. Esta divisão pode ser dividida em até seis seções, mas usualmente são utilizadas três seções (SECTIONS):

- **FILE SECTION**: seção que define a estrutura dos arquivos de dados. Esta definição envolve a descrição do arquivo e seus respectivos registros. Para cada SELECT definido temos uma definição de arquivo na FILE SECTION;
- **WORKING-STORAGE SECTION**: seção que descreve e armazena numa área de memória todos os dados, informações, variáveis e constantes, com valores definidos ou não, a serem manipulados pelo programa;
- **SCREEN SECTION**: extensão da DATA DIVISION que possui recursos para facilitar a formatação e descrição dos itens de tela e sua posterior manipulação na PROCEDURE DIVISION. Serve para: especificar a posição exata na tela de determinados campos; entrar com dados digitados em posições específicas; mostrar valores literais em posições predeterminadas; definir atributos de tela e controlar recursos de teclado. Na Figura 38 é apresentado um trecho desta divisão.

```
7   8     12
    DATA DIVISION.
    [FILE SECTION : *. *
    FD  <nome de arquivo>
          LABEL RECORD {OMMITED} * para arquivos de impressão *
                       {STANDARD} * para arquivos em disco *
          VALUE OF FILE-ID  "c:\nome-externo.extensão"  * máximo 8 caracteres no nome-externo *
          [BLOCK CONTAINS  <número-inteiro>  RECORD]
          [RECORD  CONTAINS  <número-inteiro> CHARACTERS] * soma do
          tamanho de todos os campos do registro *
          [DATA RECORD  <nome-registro>.] ]
    01    <nome-registro>.  * item de grupo *
          03 <nome-campo1>                          PIC X(<número-inteiro>).  * item elementar *
          03 <nome-campo2>.  * item de grupo *
```

```
                05 <nome-campo21>                          PIC 9(<número-inteiro>).  * item elementar *
                05 <nome-campo22>                          PIC 999999.  * item elementar *
          03 <nome-campo3>                                 PIC 9(<número-inteiro>)V(<número-inteiro>)
          03 <nome-campo4>                                 PIC 9999V99
          03 <nome-campo5>                                 PIC XXXXXXXX.
          03 <nome-campo6>                                 PIC A(<número-inteiro>).
          03 <nome-campo7>                                 PIC AAAAAAAAA.
          03 FILLER                                        PIC X(10).
*
          WORKING-STORAGE SECTION.

    77    <campo-aux>                                      PIC X(9)   [VALUE]  “COBOL”.
    01    <campo-cont>.                                    PIC 999   [VALUE] ZEROS,
                                                            ZEROES ou 0
    01    <campo-total>.                                   PIC 9999v99 [VALUE] ZEROS, ZEROES ou 0
    77    <campo-flag>                                     PIC 9     [VALUE]  0.
    01    <reg-aux>.
          03 <campo1>                                      PIC X(40).
          03 <campo2>                                      PIC 9.
                88 .<campo21>                              VALUE <valor> * pode ser assumido pelo campo
                                                           *.
                88  <campo22>                             VALUES <valor1>, <valor2>, <volorn>.
          03 <campo3>                                      PIC 9(12).
          03 <campo4> REDEFINES <campo3>.
                05 <campo41>                               PIC X(02).
                05 <campo42>                               PIC 9(08).
                05 <campo43>                               PIC X(02).
```

Figura 38 – DATA Division

4. PROCEDURE DIVISION: esta divisão contém as instruções e o curso lógico necessário para chegar-se ao resultado final. Nas Figura 39, 40, 41 e 42 são apresentados alguns comandos utilizados nesta divisão.

```
ACCEPT - COMANDO PARA PEGAR AS INFORMAÇÕES VIA TECLADO
                ACCEPT campo FROM {DAY} . FORMATO AADDD
                                          {DATE}. FORMATO AAMMDD
                                          {TIME}. FORMATO HHMMSSCC
                ACCEPT campo.
```

Figura 39 – Comando ACCEPT

```
PERFORM-EXECUTA PARTES SEPARADAS DO CORPO PRINCIPAL DO PROGRAMA.
   PERFORM {paragrafo} {seção} [TRHU {parag-2} {seção-2}]
   PERFORM {paragrafo} {seção} {inteiro} {campo} TIMES.
   PERFORM {paragrafo} {seção} UNTIL condição.
   PERFORM {paragrafo} {seção} VARYING {campo} {indexador}
                FROM  {inteiro} {indexador} {campo} BY {inteiro} {campo}  UNTIL condição
   [AFTER VARYING {indx} {campo} FROM {int} {campo} {indx} BY {int} {campo} UNTIL cond]
   [AFTER VARYING {indx} {campo} FROM {int} {campo} {indx} BY {int} {campo} UNTIL cond].
```

Figura 40 – Comando PERFORM

```
GO TO - TRANSFERE  CONTROLE DO PROGRAMA PARA O PARÁGRAFO ESPECIFICADO.
                GO TO  nome-parágrafo.
```

Figura 41 – Comando GO TO

```
MOVE - MOVE DADOS DE UMA ÁREA PARA OUTRA DA MEMÓRIA.
                MOVE {inteiro} {literal} {campo} TO  campo [campo-2,...] .
```

Figura 42 – Comando MOVE

4.2.2 Construção do Domínio COBOL

Para dar suporte à abordagem AMGraA foi construído no ST Draco-PUC, o domínio COBOL, definido pela gramática livre de contexto, *parser, prettyprinter* e o transformador COBOL definido pela biblioteca de transformação COBOLKB, que é responsável pela identificação das informações sobre o modelo de negócio da aplicação legada.

Na Figura 43 são apresentados os passos realizados para a construção do domínio COBOL.

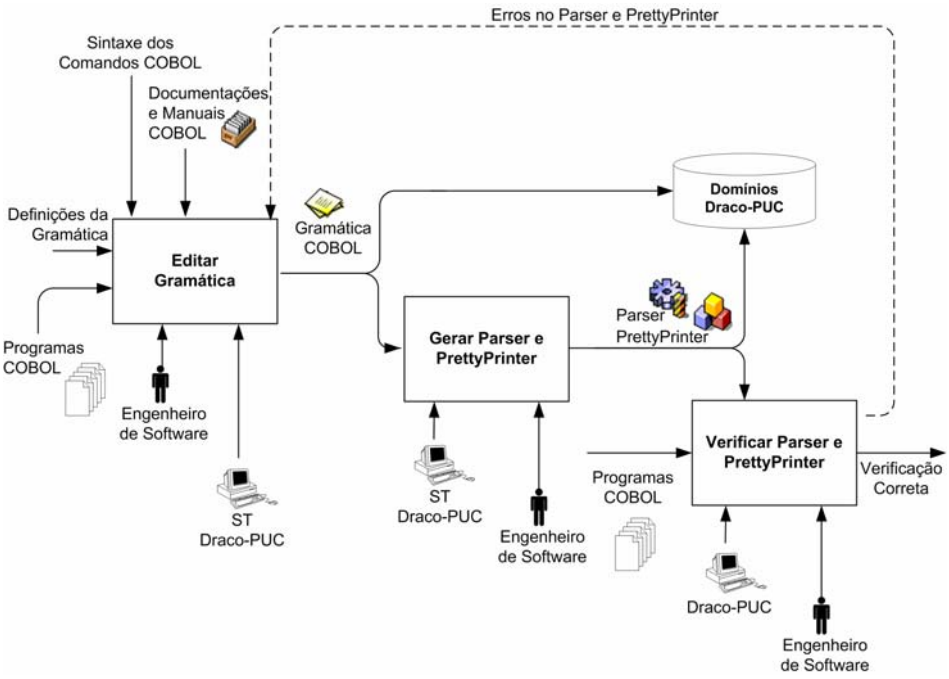


Figura 43 - Construção do domínio COBOL

A gramática COBOL da linguagem usada para implementar as aplicações legadas não é de domínio público e, portanto, não é fornecida pelos desenvolvedores da linguagem. Portanto, foi necessário fazer manualmente a engenharia reversa da gramática, a partir da sintaxe dos seus comandos e das documentações disponíveis sobre a linguagem [CARVALHO 1993], [SAADE 1997] e [SAADE 1998].

No primeiro passo, Editar Gramática, a sintaxe de cada um dos comandos da linguagem foi identificada, pela análise do código fonte de aplicações escritas em COBOL, através da consulta a documentações e manuais COBOL.

No passo Gerar Parser e PrettyPrinter, o ST Draco-PUC gera automaticamente, a partir das descrições da gramática, o *parser* e *prettyprinter* do domínio COBOL. No passo Verificar Parser e PrettyPrinter, diferentes programas fontes COBOL foram submetidos ao *parser* e ao *prettyprinter* COBOL para testar e depurar o *parser* e o respectivo *prettyprinter.*

O domínio COBOL foi construído em cerca de 300 horas, aproximadamente 10 dias.

Na Figura 44 mostra-se parte das regras de produção da gramática COBOL criada.

```
%%
/********************** Corpo da Gramática**************************/
program : identification_division .nl environment_division .nl
data_division .nl procedure_division .nl
  ;
/*-----------------------------------------------------------------
                      IDENTIFICATION DIVISION
-----------------------------------------------------------------*/
identification_division : 'IDENTIFICATION' .sp 'DIVISION' '.'
program_info_list*
   ;
program_info_list : .nl program_info_clause
                  ;
program_info_clause : 'AUTHOR' '.' info_opts
   | 'INSTALLATION' '.' info_opts
   | 'DATE-WRITTEN' '.' info_opts
   | 'DATE-COMPILED' '.' info_opts
   | 'SECURITY' '.' info_opts
   | 'PROGRAM-ID' '.' word_list* prg_options '.'
   ;
info_opts : info_option*
   ;
word_list_item : IDEN
   | INTEGER
   | DECIMAL
   | STRI
   | DATE
   ;
prg_options : (.sp prg_name_opt)?
   ;
prg_name_opt : is_opt .sp common_initial .sp optional_program
   ;
   ;

/*-----------------------------------------------------------------
                       ENVIRONMENT DIVISION
-----------------------------------------------------------------*/
environment_division  : 'ENVIRONMENT' .sp 'DIVISION' '.' .nl
config_section io_section_option
   ;
/*-----------------------------------------------------------------
                      CONFIGURATION SECTION
-----------------------------------------------------------------*/
config_section : 'CONFIGURATION' .sp 'SECTION' '.' source_comp_opt
object_comp_opt special_opt
   ;
source_comp_opt : (.nl source_computer)?
   ;
source_computer : 'SOURCE-COMPUTER' '.' .sp IDEN .sp source_debug_opt
'.'
                ;
source_debug_opt : source_debug?
   ;
source_debug : 'WITH' .sp 'DEBUGGING' .sp 'MODE'
   ;
object_comp_opt : (.nl object_computer)?
   ;
...
```

Figura 44 – Parte da gramática COBOL

4.2.3 Construção do Transformador para Identificação do Modelo de Negócio

Para dar suporte a abordagem AMGraA foi construído o transformador intradomínio COBOLKB. O Engenheiro de *Software* editou as regras de transformações com especificações que mapeiam a sintaxe e semântica da linguagem da aplicação legada COBOL.

Conforme apresentado na seção 3.2 do capítulo 3, o transformador de identificação é aplicado ao código da aplicação legada COBOL para:

- **Identificar tabelas e campos**: as tabelas e campos são identificados a partir da FILE SECTION, seção que compõe a DATA DIVISION e que define a estrutura dos arquivos de dados utilizados pela aplicação;
- **Identificar regras de negócio**: as regras de negócio são identificadas a partir dos blocos de código existentes na PROCEDURE DIVISION. Cada bloco desta divisão dá origem a uma regra de negócio. Uma SECTION ou bloco interno de uma SECTION dá origem a uma regra de negócio;
- **Identificar relacionamentos**: os relacionamentos entre as tabelas são identificados a partir dos comandos de leitura, READ. Ao identificar um comando de leitura verificam-se quais são os campos do índice primário e quais as informações atribuídas para estes campos. Se as informações atribuídas a estes campos de índices forem informações de outros arquivos ou tabelas isso indica um relacionamento entre os arquivos ou tabelas;
- **Identificar hierarquia de chamadas de programas**: a hierarquia de chamadas de programas é identificada a partir de comandos de chamada de programa, como CALL.
- **Identificar fluxo de execução da aplicação legada**: o fluxo de execução da aplicação legada é identificado pelos comandos que determinam a seqüência ou uma alteração na seqüência de execução da aplicação, tais como: GO, GOTO, PERFORM e EXIT;
- **Identificar casos de uso**: os casos de uso da aplicação são identificados manualmente através da hierarquia de chamadas. Ao identificar que programa chama qual programa, pode-se indicar o início de um caso de uso.
- **Identificar interfaces de usuário e objetos de interface**: as interfaces de usuário e os objetos de interface são identificados a partir do código existente na SCREEN SECTION. Esta seção é uma extensão da DATA DIVISION e possui recursos para facilitar a formatação e descrição dos objetos da interface.

As transformações que implementam estas atividades no transformador COBOLKB não usam padrões de substituição, e apenas armazenam fatos na base de conhecimento para identificar os dados do modelo de negócio da aplicação legada. As transformações para a identificação do modelo de negócio da aplicação legada foram agrupadas em conjuntos de transformações. Na Figura 45 são apresentados os principais conjuntos e a seqüência em que esses são aplicados.

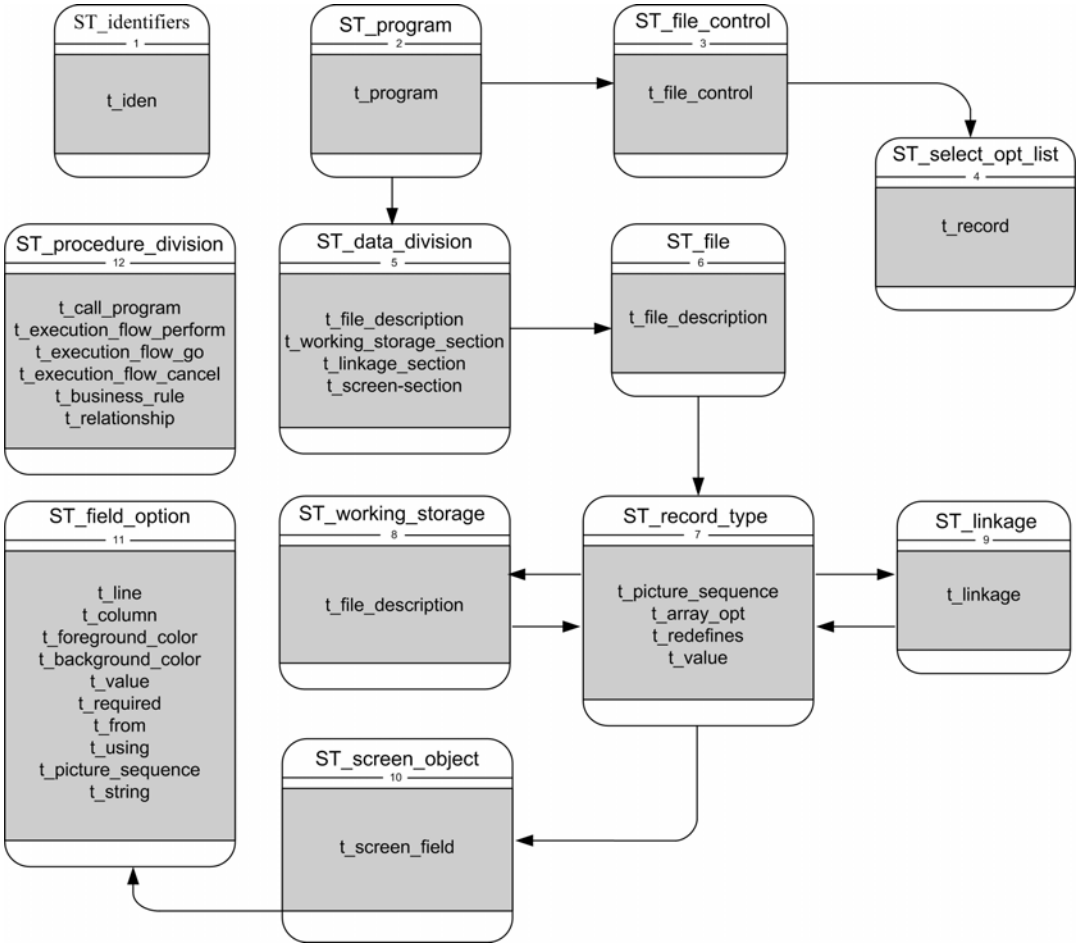


Figura 45 – Principais conjuntos de transformações para coleta de fatos

O conjunto ST_identifiers é aplicado para padronizar o nome dos identificadores. O conjunto ST_program reconhece cada unidade de programa COBOL. Esse conjunto de transformações aplica, primeiramente, o conjunto de transformações ST_file_control para reconhecer a declaração dos arquivos. Em seguida, o conjunto ST_file_control aplica o conjunto de transformações ST_select_opt_list para reconhecer os opcionais da declaração dos arquivos.

O conjunto ST_program aplica o conjunto de transformações ST_data_division, que reconhece a DATA DIVISION para a identificação dos dados a serem processados ou manipulados pelo programa. Em seguida, o conjunto ST_data_division aplica o conjunto de transformações ST_file, que reconhece os arquivos.

O reconhecimento de um arquivo na FILE SECTION pela transformação t_file_description implica no reconhecimento de uma tabela e seus campos, dando origem aos fatos "table" e "field" na base de conhecimento (KB). Na Figura 46 é apresentada a transformação t_file_description.

```
TRANSFORM t_file_description
LHS: {{dast cobol.file_desc_entry
    [[file_desc_type v_file_desc_type]]
    [[iden v_iden]]
    [[file_opt_entry* v_file_opt_entry]]
    [[file_name_entry* v_file_name_entry]] .
    [[record_entry_block* v_record_entry_block]]
...
KBAssert("table([[t_temp]],[[t_temp]],[[t_tabela]],[[t_temp1]],[[t_temp2]])");
}}
TRANSFORM t_file_description
LHS: {{dast cobol.record_entry_block
    [[INTEGER v_integer]]
    [[level_name v_level_name]]
    [[record_type v_record_type]] .
...
KBAssert("field([[t_tabela]],[[t_temp1]],[[t_nome_objeto]], [[t_desc_objeto]],
[[t_desc_objeto]],[[t_formato]],[[t_mascara]],[[t_valores]],yes,[[t_ajuda]],
[[t_desc_objeto]])");
}}
```

Figura 46 - Transformação t_file_description para reconhecimento de tabelas e seus campos

Em seguida, o conjunto de transformações ST_record_type, ST_working_storage e ST_linkage são aplicados para reconhecer as propriedades dos campos, variáveis e parâmetros.

A transformação t_screen_section reconhece a SCREEN SECTION, seção que contém a descrição da interface da aplicação. O conjunto de transformação ST_screen_object é aplicado para obter dados dos itens da interface. Na Figura 47 são apresentadas as transformações t_screen_section, que reconhece uma interface e t_screen_field, que reconhece os objetos de interface, dando origem aos fatos "form" e "screenObject" na base de conhecimento.

```
TRANSFORM t_screen_section
LHS: {{dast cobol.screen_section SCREEN SECTION .
    [[screen_field_list* v_screen_field_list]]
...
KBAssertIfNew("form([[t_programa]],1,[[t_nome_tela]],[[t_tipo_interface]],[[t_nome_tela]]");
}}
TRANSFORM t_screen_field
LHS: {{dast cobol.screen_field
    [[INTEGER v_integer]]
    [[field_name_opt v_field_name_opt]]
    [[field_def_list* v_field_def_list]] .
...
KBAssert("screenObject([[t_programa]],[[t_nome_tela]],[[t_seq_item_tela]],[[t_objeto_pai]],[t_nome_o
bjeto]],[[t_desc_objeto]],[[t_tipo_objeto]],[[t_pos_linha]],[[t_pos_coluna]],[[t_altura]],[[t_largur
a]],[[t_tam_fonte]],[[t_cor_fonte]],[[t_cor_fundo]],[[t_formato]],[[t_valores]],[[t_ajuda]],[[t_masc
ara]])");
}}
```

Figura 47 – Transformações t_screen_section e t_screen_field para reconhecimento das interfaces e objetos de interface

O conjunto ST_procedure_division reconhece a PROCEDURE DIVISION, que contém as instruções e o curso lógico da aplicação. A transformação t_call_program reconhece, através do comando CALL, as chamadas de programas da aplicação, conforme apresentado na Figura 48.

```
TRANSFORM t_call_program
LHS: {{ dast cobol.call_stm CALL
     [[system_option v_system_option]]
     [[call_list v_call_list]]
     [[using_option v_using_option]]
     [[giving_option v_giving_option]]
     [[exception_opt v_exception_opt]]
     [[end_call_opt v_end_call_opt]]
...
KBAssert("call([[t_programName]],[[t_seq]],[[t_programCalled]])");
}}
```

Figura 48 - Transformação t_call_program para reconhecimento das chamadas de programas

As transformações t_execution_flow_perform, t_execution_flow_go e t_execution_flow_cancel reconhecem o fluxo de execução da aplicação através dos comandos PERFORM, GO e CANCEL, respectivamente, conforme apresentado na Figura 49.

```
TRANSFORM t_execution_flow_perform
LHS: {{ dast cobol.perform_stm PERFORM
     [[iden v_iden]]
     [[thru_opt* v_thru_opt]]
     [[perform_options v_perform_options]]
     [[end_perform_opt v_end_perform_opt]]

KBAssert("executionFlow([[t_program]],[[t_command]],[[t_programCalled]],
[[t_temp1]],[[t_temp2]])");
}}

TRANSFORM t_execution_flow_go
LHS: {{ dast cobol.misc_stm GO
     [[to_option v_to_option]]
     [[variables v_variables]]
     [[depending_option v_dependig_option]]
...
KBAssert("executionFlow([[t_program]],[[t_command]],[[t_programCalled]],
[[t_temp1]],[[t_temp2]])");
}}

TRANSFORM t_execution_flow_cancel
LHS: {{ dast cobol.misc_stm CANCEL value
     [[EXIT]]
     [[EXIT PROGRAMA]]
     [[EXIT PERFORM]]
...
KBAssert("executionFlow([[t_program]],[[t_command]],[[t_programCalled]],
[[t_temp1]],[[t_temp2]])");
}}
```

Figura 49 – Transformações t_execution_flow_perform, t_execution_flow_go e t_execution_flow_cancel para o reconhecimento do fluxo de execução da aplicação

A transformação t_business_rule reconhece as regras de negócio da aplicação através dos blocos de código existentes na PROCEDURE DIVISION. Uma SECTION ou bloco interno de uma SECTION vai dar origem a uma regra de negócio. Na Figura 50 é apresentada a transformação t_business_rule.

```
TRANSFORM t_business_rule
LHS: {{ dast cobol.body_structs
     [[iden v_inden]] SECTION .
     [[statements v_statements]]
     [[label v_label]]. [[statements v_statements]]
     [[copy_block v_copy_block]];
...
KBAssert("businessRule([[t_programName]],[[t_sectionName]],
[[t_businessRuleSeq]])");
}}
```

Figura 50 - Transformação t_business_rule para reconhecimento das regras de negócio da aplicação

Ainda na PROCEDURE DIVISION, podem-se reconhecer os relacionamentos entre as tabelas identificadas. Na Figura 51 tem-se a transformação t_relationship, que reconhece um relacionamento através dos comandos de leitura e verificação de informações atribuídas aos campos índices.

```
TRANSFORM t_relationship
LHS: {{ dast cobol.read_stm READ
     [[iden v_inden]]
     [[read_type v_read_type]]
     [[record_option v_record_option]]
     [[into_clause v_into_clause]]
     [[invalid_option v_invalid_option]]
     [[end_read_opt end_read_opt]]
...
KBAssert("relationship([[t_originTableName]],[[t_relationshipSeq]],
[[t_originTableId]],                    [[t_originCardinality]],[[t_destinyTableName]],
[[t_destinyTableId]],[[t_destinyCardinality]])");
}}
```

Figura 51 – Transformação t_relationship para reconhecimento dos relacionamentos entre as tabelas

Na Tabela 4 são apresentados fatos armazenados na base de conhecimento, durante a identificação dos dados do modelo de negócio pelo transformador COBOLKB, destacando-se os fatos sobre a estrutura da base de dados, as interfaces e objetos de interface, e a hierarquia de chamada das unidades de programa. Como exemplos, para cada interface identificada no código fonte, tem-se o fato "form", com as informações programName, que identifica o nome do programa da interface, formSeq, que identifica um contador de interfaces identificadas, formName, que identifica o nome da interface, interfaceType, que descreve o tipo da interface, podendo ser, simples, master/detail, grid ou associativa, e title, que descreve o título da interface.

Para cada declaração de variável na WORKING-STORAGE SECTION do código fonte, tem-se o fato "variable", com as informações programa, que identifica o nome do programa origem, unidadePrograma, que identifica o nome da unidade de programa que foi declarada a variável, e Tipo, que descreve o tipo da variável, podendo ser inteiro, decimal, caractere, lógico, etc.

Tabela 4 – Tabela de fatos armazenados na base de conhecimento (KB)

Nome do Fato	Informações
table	tableSeq, tableId, tableName, tableDesc, database
field	tableName, fieldSeq, fieldName, fieldLabel, fieldColumn, datatype, format, initialValue, mandatory, help, descriptionField
relationship	originTable, relationshipSeq, originTableId, originCardinality, destinyTable, destinyTableId, destinyCardinality
index	tableName, indexSeq, indexName
indexItem	tableName, nameIndex, indexItemSeq, field, single
form	programName, formSeq, formName, interfaceType, title
screenObject	programName, objectSeq, formName, mainObject, objectName, objectDesc, objectType, linePos, columnPos, height, width, fontHeight, fontColor, backgroundColor, format, values, help, mask
call	programName, programUnit, calledUnity
executionFlow	programName, command, programNameCalled, seq
businessRule	programName, businessRuleName, businessRuleSeq
program	programName, title
programTable	programName, seq, tableName
programUnit	programName, programUnit, unitType
message	programName, programUnit, messageSeq, messageDesc
variable	programName, programUnit, variable, type
fieldReference	programName, programUnit, table, field
fieldSave	programName, programUnit, table, field
fieldRead	programName, programUnit, table, field
createTable	programName, programUnit, accessSeq, table
deleteTable	programName, programUnit, accessSeq, table
updateTable	programName, procName, seq, tableName
readTable	programName, programUnit, accessSeq, table
ItemRead	programName, programUnit, accessSeq, table, readSeq, field
useCase	seq, elementoRecuperadoSeq, useCaseName
useCaseActor	useCaseActor, useCaseName
useCaseProg	useCaseName, seq, programName

A criação do domínio COBOL e do transformador levou cerca de 1500 horas, aproximadamente 2 meses e 2 dias, considerando os testes com a passagem dos programas. A criação deste domínio foi realizada pelo Participante 2, com a colaboração do Participante 1 e 4. É importante considerar também o conhecimento e a experiência do Participante 2 na utilização do ST Draco-PUC. O Participante 4 informava aos Participantes 1 e 2 as informações a serem recuperadas e a estrutura das descrições do modelo de negócio da aplicação a ser seguida e refletida no arquivo XML gerado.

O domínio COBOL criado reconhece aplicações COBOL escritas pelos fabricantes Micro Focus e Fujitsu. O transformador pode ser adaptado para atender a outras variações da linguagem COBOL. Uma adaptação para outra versão do COBOL implica em alterar o seu domínio e transformador, o que exigiria cerca de 500 horas, aproximadamente 21 dias. Este número é baseado no histórico de outros projetos de Reengenharia de *Software*.

4.2.4 Construção do Transformador para Descrever os Fatos em XML

Para descrever as informações do modelo de negócio armazenadas na Base de Conhecimento (KB) do ST Draco-PUC em XML e dar suporte à abordagem AMGraA, permitindo a integração entre as ferramentas utilizadas na abordagem, foi construído o transformador DracoKBXML.

A construção dos conjuntos de transformações para este transformador é análogo ao apresentado na seção 4.2.3. As transformações lêem os fatos armazenados e geram o XML com as descrições.

Na Figura 52 é apresentada a leitura das informações armazenadas para o fato "table" na Base de Conhecimento e a geração do XML, através do template T_xml, que segue a DTD MPPImportacao para a geração do arquivo XML.

```
sprintf(t_query, "table(%s, *b, *c, *d, *e)", t_temp);
     while (KBSolve(t_query))    {
         t_attrib_tot    =    7;

         /* Lê a informação da base de conhecimento e move para variáveis */
         sprintf(t_tabela, "%s", KBRetrieve("*c", 1));
         sprintf(t_attrib_value[1], "%s", "");
         sprintf(t_attrib_name[1],  "%s", "codigoSistema");
         sprintf(t_attrib_value[2], "%s", "");
         sprintf(t_attrib_name[2],  "%s", "codigoProjeto");
         sprintf(t_attrib_value[3], "%s", t_tabela);
         sprintf(t_attrib_name[3],  "%s", "nomeTabela");
         sprintf(t_attrib_value[4], "%s", t_temp2);
         sprintf(t_attrib_name[4],  "%s", "descricaoTabela");
         sprintf(t_attrib_value[5], "%s", "");
         sprintf(t_attrib_name[5],  "%s", "flgView");
         sprintf(t_attrib_value[6], "%s", "");
         sprintf(t_attrib_name[6],  "%s", "flgController");

         #include "lerAtributos.i"
...
         t_cont++;
         sprintf(t_temp, "%d", t_cont);
         sprintf(t_query, "table(%s, *b, *

         #include "geraFato.i"
     }

     /*** Move a estrutura completa dos fatos para o arquivo XML ***/
     TEMPLATE("T_xml")
         MOVE1(WS_attribute_list, "WS_attribute_list");
         PLACE_AT(WS_tabela);
     END_TEMPLATE;
```

```
TEMPLATE T_xml
RHS:{{dast xml.document
 <?xml version="1.0" encoding="iso-8859-1"?>
 <!DOCTYPE MPPImportacao SYSTEM "MPPImportacao.dtd">
 [[body_element* WS_attribute_list]]
}}
```

Figura 52 – Transformador para descrever os fatos da Base de Conhecimento (KB) em XML

Embora o transformador DracoKBXML tenha sido construído no contexto de um projeto para a modernização de uma aplicação COBOL, sua utilização é independente de linguagem, podendo ser utilizado futuramente em outros projetos.

4.3 Analisar Aplicação Legada

Nesta fase o Engenheiro de *Software*, representado pelo Participante 4, faz a análise da aplicação legada, ou seja, dos módulos selecionados, aplicando o transformador de identificação apresentado na seção 4.2, construído no ST Draco-PUC. As transformações definidas identificam dados do modelo de negócio da aplicação, armazenando fatos na base de conhecimento.

Primeiramente, reunem-se informações sobre a aplicação legada, neste estudo de caso, somente o código fonte da aplicação. Em seguida, foram selecionados os módulos Básico, Vendas e Materiais para serem submetidos ao ST Draco-PUC, que aplicou as transformações construídas na fase anterior para a extração dos dados sobre o modelo de negócio da aplicação.

O transformador analisou a aplicação legada seguindo suas divisões, primeiramente, a IDENTIFICATION DIVISION para coletar dados sobre a autoria da aplicação, em seguida, a DATA DIVISION, e suas seções FILE SECTION, WORKING-STORAGE SECTION e SCREEN SECTION.

Na Figura 53 é apresentada a transformação t_file_description sendo aplicada ao programa "ven010". A transformação reconhece a FILE SECTION, que contém as "FDs", que possuem a estrutura dos arquivos de dados, para a identificação das tabelas e seus campos.

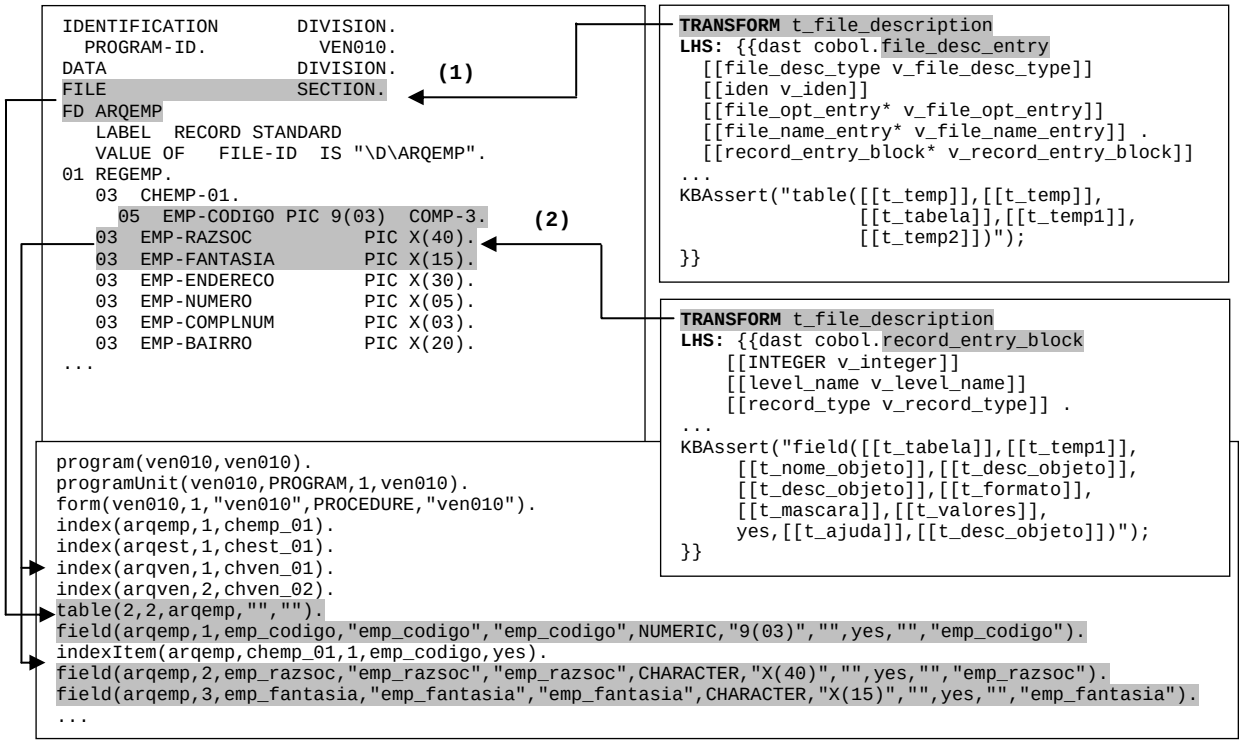


Figura 53 - Identificação da tabela "ARQEMP" e seus campos

Em (1) é identificado o arquivo ARQEMP dando origem a um fato "table" na base de conhecimento. Para as tabelas são armazenados um contador de tabelas, o índice da tabela no banco de dados, o nome da tabela, sua descrição e o nome do banco, no caso de uma aplicação orientada a banco de dados. Em (2) os campos emp_codigo, emp_raszoc e emp_fantasia, e o índices da tabela, chemp_01, são identificados dando origem aos fatos "field" e "index". Para os campos são armazenados o nome da tabela, um identificador da tabela, o nome do campo, seu label, tipo de dado, formato, valor inicial, mandatário, help e a descrição do campo.

Na WORKING-STORAGE SECTION foram identificadas as variáveis do código legado da aplicação. Conforme apresentado na Figura 54, em (1) tem-se a transformação t_working_storage_section que reconhece a seção WORKING-STORAGE, e em (2) a transformação t_file_description identifica as variáveis ws-tela1, ws-tell, ws-opc1 e ws-impressora, dando origem aos fatos "object" e "objectDetail" do programa "ven010".

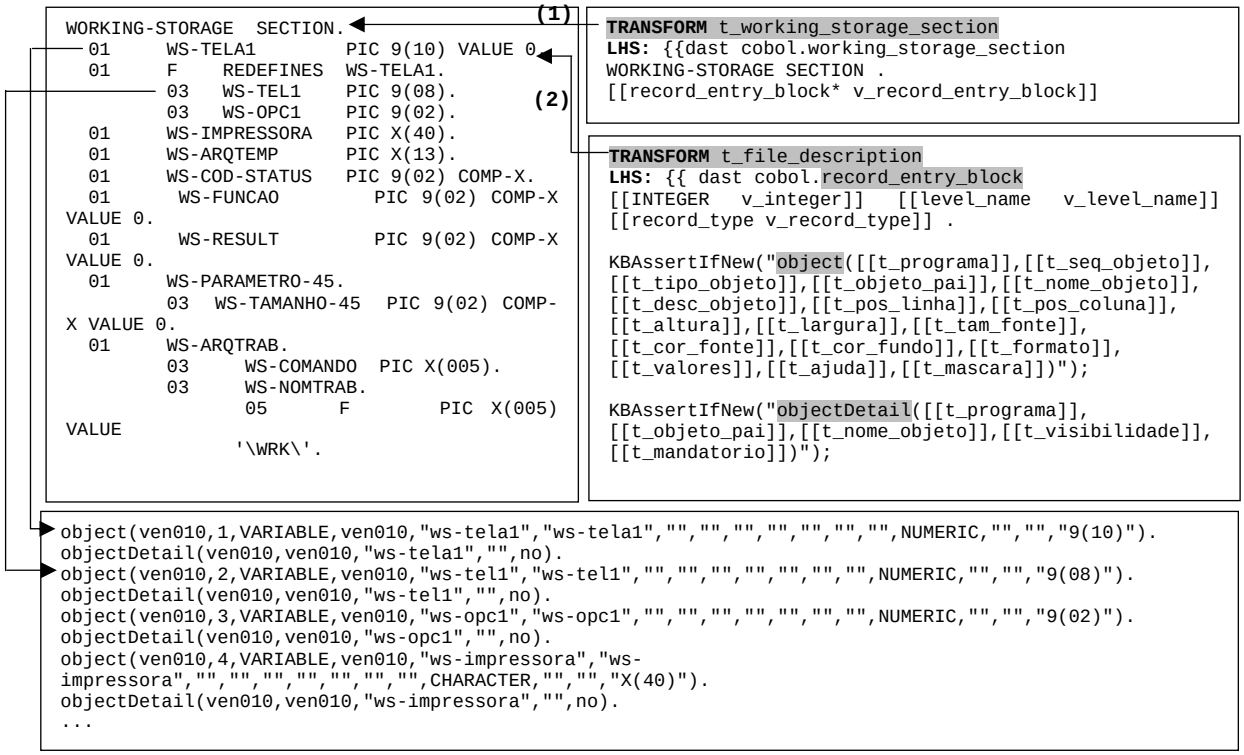


Figura 54 – Identificação da WORKING-STORAGE SECTION e variáveis do programa

Na SCREEN SECTION foram identificadas as interfaces e objetos de interface da aplicação, conforme apresentado na Figura 55. Em (1) tem-se a transformação t_screen_section, que reconhece a SCREEN SECTION, identificando uma interface e dando origem ao fato "form" (1.1) na base de conhecimento (KB), armazenando o nome do programa, o nome, tipo e título da interface.

Em (2) a transformação t_screen_field reconhece os objetos de uma interface, dando origem aos fatos "screenObject" com informações de cada objeto. Em (2.1), (2.2) e (2.3) tem-se o reconhecimento dos objetos "emp-razsoc", "ws-nomtel" e "label1". Para os objetos de interface são armazenados o nome do programa, um identificador de objetos, o nome da interface, o nome do objeto identificado, sua descrição, tipo, linha e coluna de posicionamento na tela, altura e largura do objeto, cor da fonte, cor do fundo, formato, valores e máscaras.

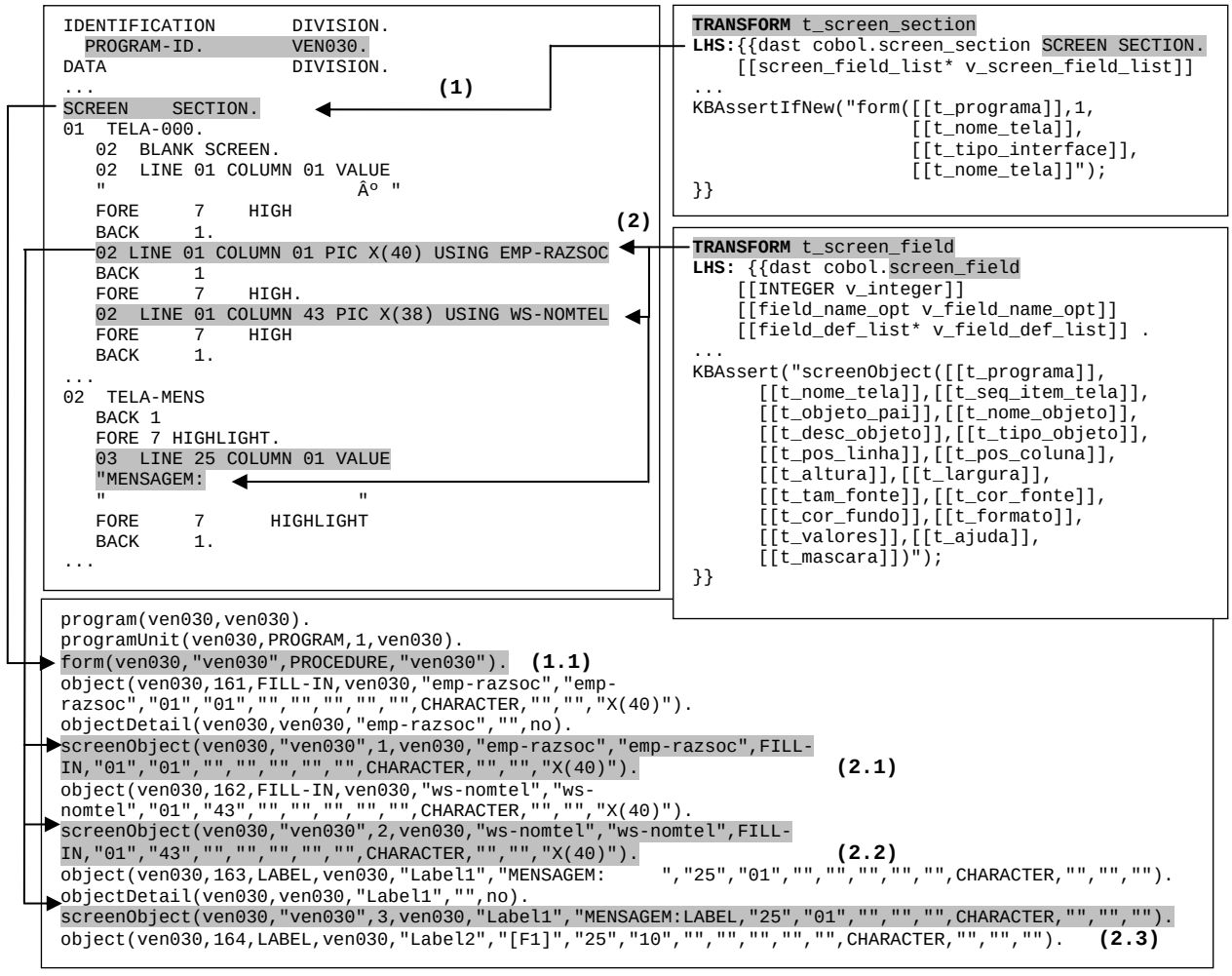


Figura 55 - Identificação da interface do programa "ven030" e seus objetos de interface

Na PROCEDURE DIVISION são identificadas as regras de negócio da aplicação como, por exemplo, do programa "ven010", conforme apresentado na Figura 56. Em (1) tem-se a transformação t_business_rule que reconhece a PROCEDURE DIVISION. Em (2) tem-se a identificação da seção B1-CONSISTE-ESTADO SECTION, dando origem a uma regra de negócio da aplicação e ao fato "business_rule" na base de conhecimento. Em (3) a SECTION B2-BUSCA-CODIGO possui um bloco interno, B2, dando origem a duas regras de negócio B2-BUSCA-CODIGO e B2. Para as regras de negócio são armazenados o nome do programa, o nome da regra de negócio e um identificador de regras de negócio.

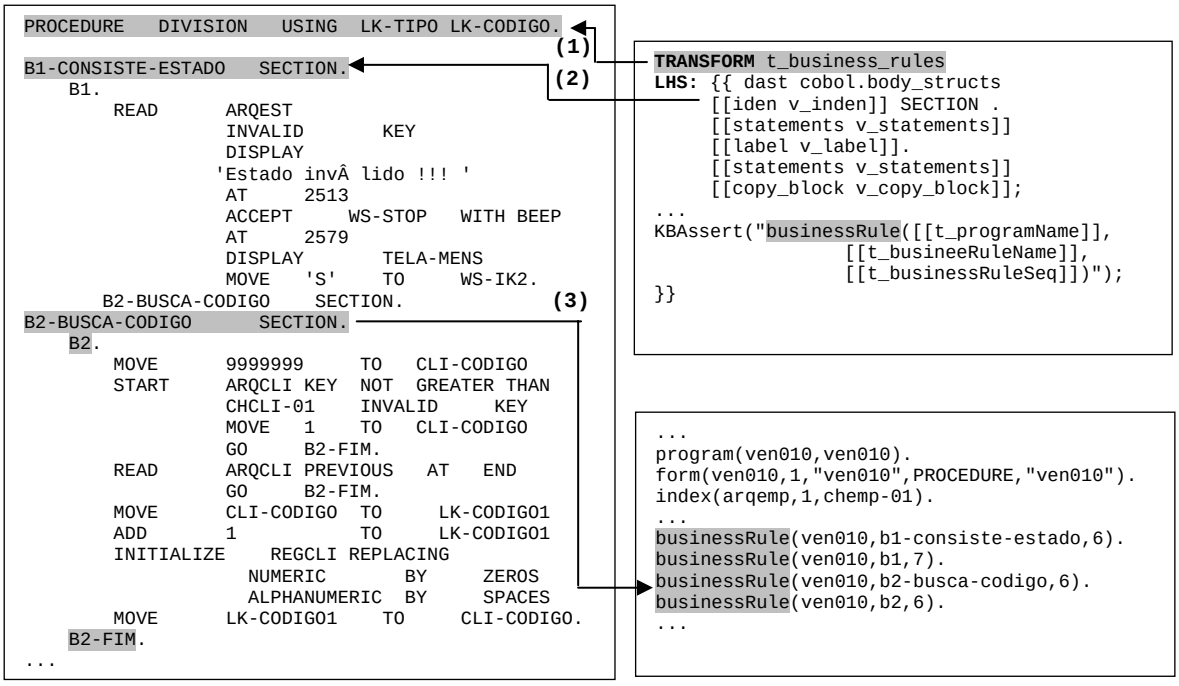


Figura 56 – Identificação da PROCEDURE DIVISION e regras de negócio

Ainda na PROCEDURE DIVISION são identificadas as chamadas de programas. Na Figura 57 é apresentada a transformação t_call_program e a identificação de duas chamadas do programa ven010. Em (1) tem-se a chamada ao programa ven011 e em (2) a chamada ao programa ven008, dando origem aos fatos "call" na base de conhecimento (3).

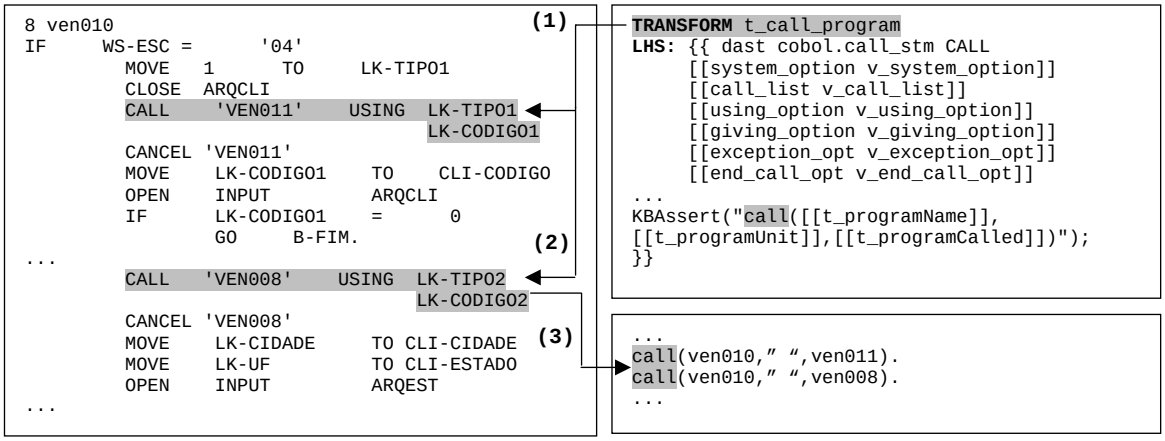


Figura 57 – Identificação das chamadas de programas

Na Figura 58 é mostrada a identificação do fluxo de execução da aplicação, através da análise do programa "ven015". O fluxo é identificado por comandos que determinam como será a execução da aplicação. O fluxo pode ser linear ou não-linear, dependendo dos comandos encontrados no código. Em (1) a transformação t_execution_flow_perform reconhece o comando PERFORM que desvia a execução da aplicação para executar uma rotina e retorna ao ponto de

chamada após a execução da rotina. Em (2) a transformação t_execution_flow_go reconhece o comando GO que desvia a execução para executar uma rotina, não retornando ao ponto de chamada no final da execução. Em (3) a transformação t_execution_flow_cancel reconhece o comando CANCEL que finaliza a execução de uma rotina. As transformações dão origem ao fato "executionFlow" na base de conhecimento.

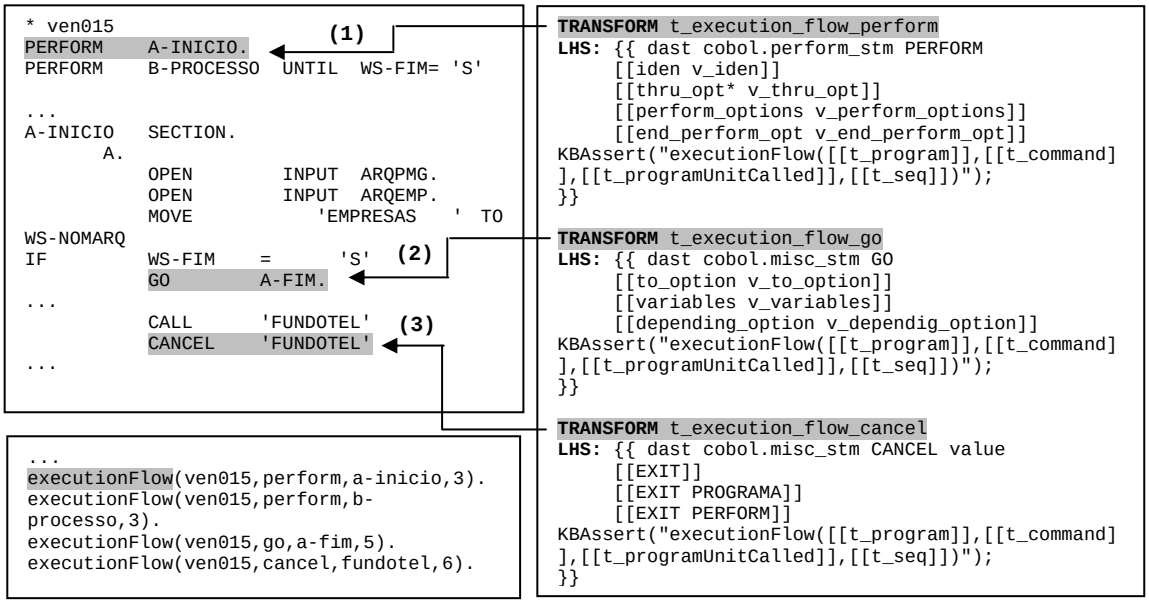


Figura 58 – Identificação do fluxo de execução da aplicação

A transformação t_relationship reconhece um relacionamento através do comando de leitura READ. Na Figura 59 é mostrada a identificação do relacionamento entre as tabelas ARQCLI e ARQEST, reconhecido através da atribuição de informações da tabela ARQCLI para os campos de índices da tabela ARQEST (1) e da leitura dessas informações (2).

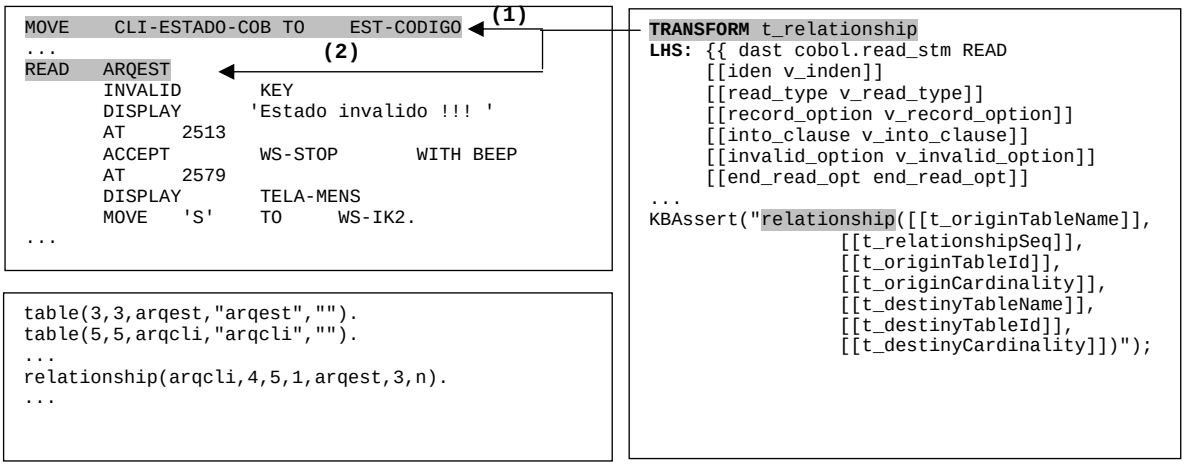


Figura 59 – Identificação de relacionamentos

Os casos de uso da aplicação são identificados manualmente pelo Engenheiro de *Software* através da hierarquia de chamadas. O Engenheiro de *Software*, ao identificar as chamadas de

programas, especifica qual o início de um caso de uso. Para os casos de uso são armazenados os fatos "useCase". Cada um desses fatos possui um identificador e o nome do caso de uso. Também são recuperadas informações sobre o ator do caso de uso através do fato "useCaseActor" e do programa através do fato "useCaseProg".

Na Figura 60 são mostrados os casos de uso identificados pelo Engenheiro de *Software*.

```
actor(1,2013,"Administrativo").
useCase(1,149,"ven010").
useCaseActor("ven010",1,149,"Administrativo",2013).
useCaseProg("ven010",1,"ven010").
useCase(2,150,"ven015").
useCaseActor("ven015",1,150,"Administrativo",2013).
useCaseProg("ven015",1,"ven015").
useCase(3,151,"ven030").
useCaseActor("ven030",1,151,"Administrativo",2013).
useCaseProg("ven030",1,"ven030").
```

Figura 60 – Identificação de casos de uso

Na Figura 61 é apresentada a Base de Conhecimento com alguns fatos coletados após a análise dos módulos: Básico, Vendas e Financeiro.

```
actor(1,2013,"Administrativo").
useCase(1,149,"ven010").
useCaseActor("ven010",1,149,"Administrativo",2013).
useCaseProg("ven010",1,"ven010").
useCase(2,150,"ven015").
firstProgram(ven010).
program(ven010,ven010).
form(ven010,1,"ven010",PROCEDURE,"ven010").
index(arqemp,1,chemp-01).
index(arqest,1,chest-01).
index(arqven,1,chven-01).
index(arqven,2,chven-02).
index(arqcli,1,chcli-01).
...
table(1,1,arqpmg,"arqpmg","").
field(arqpmg,1,pmg-razsoc,"pmg-razsoc","pmg-razsoc",CHARACTER,"X(40)","",yes,"","pmg-razsoc").
field(arqpmg,2,pmg-endereco,"pmg-endereco","pmg-endereco",CHARACTER,"X(40)","",yes,"","pmg-endereco").
field(arqpmg,3,pmg-bairro,"pmg-bairro","pmg-bairro",CHARACTER,"X(20)","",yes,"","pmg-bairro").
...
table(2,2,arqemp,"arqemp","").
field(arqemp,1,emp-codigo,"emp-codigo","emp-codigo",NUMERIC,"9(03)","",yes,"","emp-codigo").
indexItem(arqemp,chemp-01,1,emp-codigo,yes).
field(arqemp,2,emp-razsoc,"emp-razsoc","emp-razsoc",CHARACTER,"X(40)","",yes,"","emp-razsoc").
programTable(ven010,2,arqemp).
table(3,3,arqest,"arqest","").
programTable(ven010,3,arqest).
table(4,4,arqven,"arqven","").
object(ven010,1,VARIABLE,ven010,"ws-tela1","ws-tela1","","","","","","","",NUMERIC,"","","9(10)").
objectDetail(ven010,ven010,"ws-tela1","",no).
object(ven010,2,VARIABLE,ven010,"ws-tel1","ws-tel1","","","","","","","",NUMERIC,"","","9(08)").
objectDetail(ven010,ven010,"ws-tel1","",no).
dataRedefine(ven010,ws-tela1,ws-tel1).
object(ven010,3,VARIABLE,ven010,"ws-opc1","ws-opc1","","","","","","","",NUMERIC,"","","9(02)").
object(ven010,145,LABEL,ven010,"Label28","PESSOA:","17","12","","","","","",CHARACTER,"","","").
objectDetail(ven010,ven010,"Label28","",no).
screenObject(ven010,"ven010",31,ven010,"Label28","PESSOA:",LABEL,"17","12","","","","","",CHARACTER,"",
"","").
screenObject(ven030,"ven030",49,ven030,"pdv-idetra","pdv-idetra",FILL-
programUnit(ven010,PROGRAM,1,ven010).
programUnit(ven010,PROCEDURE,2,ven010p).
programUnit(ven010,PROCEDURE,3,a-inicio).
programUnit(ven010,PROCEDURE,4,A).
programUnit(ven015,PROGRAM,1,ven015).
programUnit(ven015,PROCEDURE,2,ven005p).
programUnit(ven015,PROCEDURE,3,a-inicio).
programUnit(ven015,PROCEDURE,4,A).
programUnit(ven030,PROCEDURE,32,ler-cond-pagto-vl).
```

Figura 61 – Base de Conhecimento com fatos coletados

A fase de Análise da Aplicação Legada levou cerca de 30 minutos, considerando a análise nos 3 módulos selecionados.

Após a análise do código da aplicação, as informações do modelo de negócio, armazenados na Base de Conhecimento do ST Draco-PUC, são descritos no formato XML através da aplicação das transformações do transformador DracoKBXML.

Na Figura 62 é apresentada, à esquerda, a DTD (Document Type Definition), MPPImportacao. Esta DTD foi definida para a validação do arquivo XML gerado para importação das informações do modelo de negócio e descreve as regras que o formato XML deve seguir. À direita tem-se parte de um arquivo XML gerado.

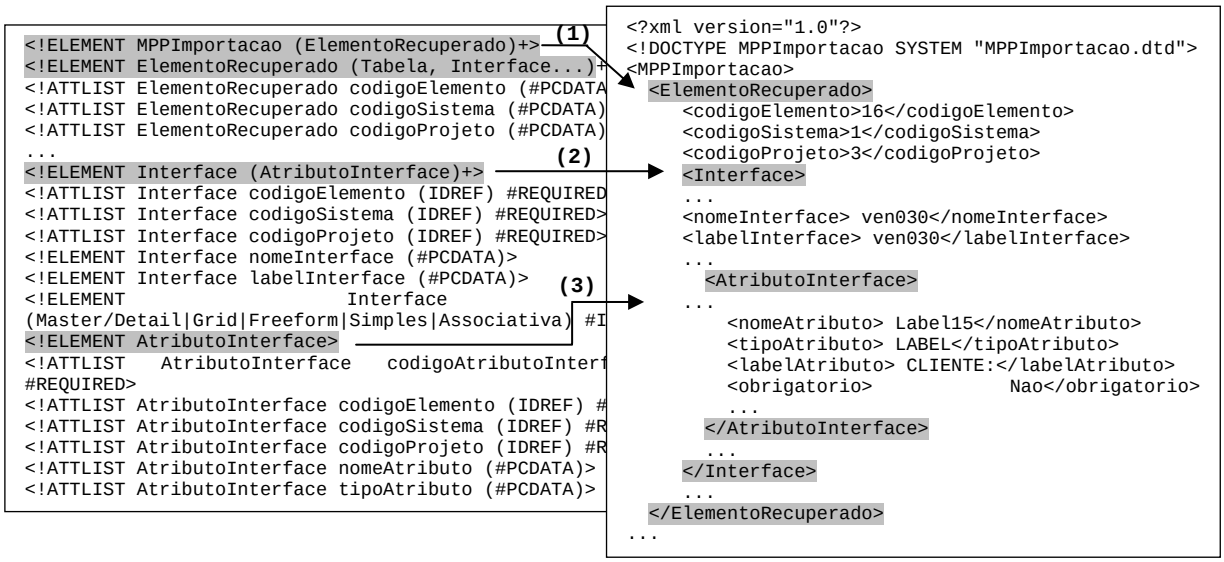


Figura 62 – Descrições XML do modelo de negócio da aplicação legada

De acordo com a Figura 62, pode haver uma ou várias tags <ElementoRecuperado> (1), sendo que esta tag pode ser composta por uma ou mais *tags* <Tabela> e <Interface>. A *tag* <Interface> pode ser composta por uma ou mais tags <AtributoInterface> (2). As descrições das interfaces e dos objetos de interface da aplicação estão armazenadas nas tags <Interface> </Interface> e <AtributoInterface> </AtributoInterface>, respectivamente.

O arquivo XML foi importado na ferramenta MPP, onde o Engenheiro de *Software*, representado nesta fase pelos Participantes 3 e 4, fizeram o planejamento do projeto de migração.

4.4 Planejar Projeto de Migração

Nesta fase os Participantes 3 e 4 fizeram o planejamento do projeto de migração da aplicação legada na ferramenta Migration Project Planning (MPP).

Primeiramente, foi criado um projeto para a migração da aplicação, e em seguida importado as descrições XML do modelo de negócio da aplicação legada, obtidas na fase anterior.

Com os dados importados, pode-se analisar as informações coletadas pelo ST Draco-PUC, fazendo alterações como a modificação dos nomes de alguns programas e nome de campos e *labels* para nomes mais significativos.

Em seguida, iniciou-se o planejamento do projeto de migração da aplicação Ômega. Foram definidas as etapas do projeto, os módulos e seus programas a serem migrados em cada etapa, a data do planejamento, o início e o fim da execução de cada etapa.

Foram definidas também as camadas e as tecnologias envolvidas na migração. O Engenheiro especifica como será realizada a comunicação dos módulos e programas migrados para cada cenário definido pelas etapas. Dentre as etapas definidas para o projeto de migração da aplicação Ômega destacam-se 5 etapas:

1. **Etapa de Migração de Pedidos para *Web***: esta etapa tem como objetivo disponibilizar as funcionalidades referentes ao programa "ven030" na Web;
2. **Etapa de Disponibilização de serviços através de *Web Service* para permitir que a nova aplicação Pedidos acesse a aplicação legada**: esta etapa visa a elaboração de um *Web Service* para permitir a comunicação com a aplicação legada.
3. **Etapa de Alteração de Clientes e Vendedores para possibilitar seu funcionamento em modo "batch"**: esta etapa visa alterar os programas "ven010" e "ven015" para que possam aceitar parâmetros e funcionar em modo "batch";
4. **Etapa de Migração de Clientes**: esta etapa visa migrar as interfaces de usuário do programa "ven010" e possibilitar o armazenamento dos dados de clientes em banco de dados; e
5. **Etapa de Migração de Vendedores**: esta etapa tem como objetivo migrar as interfaces de usuário do programa "ven015" e possibilitar o armazenamento dos dados de vendedores em banco de dados.

Dentre as etapas definidas, as etapas 2 e 3 permitem a comunicação da parte migrada com a parte legada da aplicação.

Após a definição das etapas, especificaram-se as prioridades das etapas de migração e nesta seqüência foram realizadas as exportações de suas informações para o Apyon Studio. A exportação é realizada através do arquivo XML que contém informações do planejamento da etapa e dos módulos e programas envolvidos.

Na Figura 63 é mostrado, por exemplo, à esquerda, a definição da etapa "Migração de Pedidos para *Web*", onde o programa "ven030" será migrado para a *Web*. À direita tem-se a especificação da camada de Interface para *Web* e tecnologia ASPX utilizada na migração.

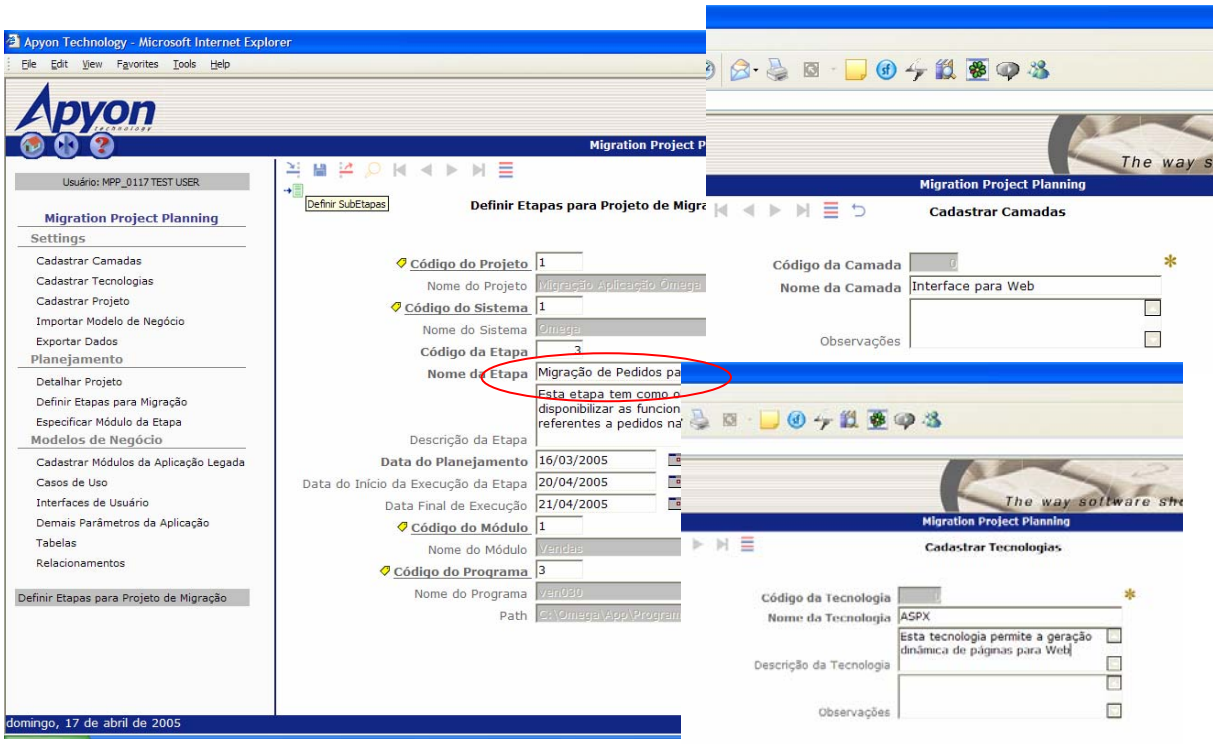


Figura 63 - Planejamento da etapa "Migração de Pedidos para Web" na ferramenta MPP

Na Figura 64 é mostrada a exportação das especificações da etapa "Migração de Pedidos para *Web*" para XML.

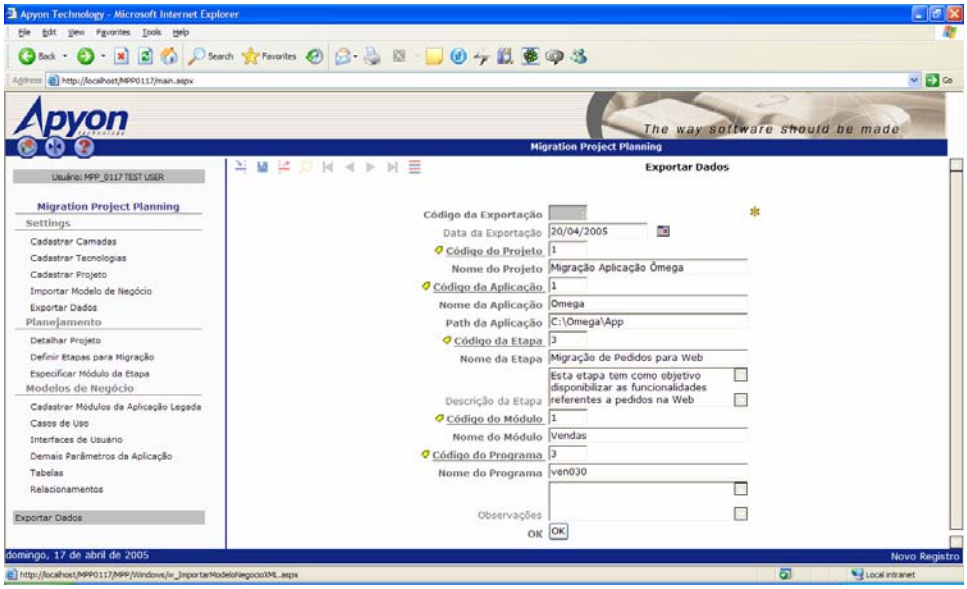


Figura 64 – Exportação das especificações da etapa "Migração de Pedidos para Web"

O arquivo XML gerado foi importado no Apyon Studio para a geração de código. Na Figura 65 é mostrado, à esquerda, parte da DTD MPPExportacao definida para a validação do arquivo XML gerado para a exportação das descrições da etapa a ser migrada. À direita tem-se parte de um arquivo no formato XML

A DTD definida é fixa para todos os projetos de migração, as especificações do arquivo XML é que variam para cada projeto.

```
<!ELEMENT MPPExportacao (Etapa)>
<!ELEMENT Etapa (Modulo)>
<!ATTLIST Etapa codigoEtapa (#PCDATA) #REQUIR
...
<!ELEMENT Modulo (Programa)+>
<!ATTLIST Modulo codigoModulo (#PCDATA) #REQU
...
<!ELEMENT Programa (ElementoRecuperado)+>
<!ATTLIST Programa codigoPrograma (#PCDATA) #
...
<!ELEMENT   ElementoRecuperado   (Tabela,
AtributoInterface, Relacionamento, CasoUso, R
<!ATTLIST ElementoRecuperado codigoElemento (
<!ATTLIST ElementoRecuperado codigoSistema (#
<!ATTLIST ElementoRecuperado codigoProjeto (#
...
<!ELEMENT Tabela (Atributo)+>
<!ATTLIST Tabela codigoElemento (IDREF) #REQU
<!ATTLIST Tabela codigoSistema (IDREF) #REQUI
...
<!ELEMENT Atributo>
<!ATTLIST Atributo codigoElemento (IDREF) #RE
<!ATTLIST Atributo codigoSistema (IDREF) #REQ
..
<!ELEMENT Interface (AtributoInterface)+>
<!ATTLIST Interface codigoElemento (IDREF) #R
<!ATTLIST Interface codigoSistema (IDREF) #RE
...
<!ELEMENT AtributoInterface>
<!ATTLIST    AtributoInterface    codigoAtrib
#REQUIRED>
<!ATTLIST AtributoInterface codigoElemento (I
...
```

```
<?xml version="1.0"?>
<!DOCTYPE MPPImportacao SYSTEM "MPPImportacao.dtd">
<MPPExportacao>
<Etapa>
     <codigoEtapa>3</codigoEtapa>
     <codigoSistema>1</codigoSistema>
     ...
   <Modulo>
     <codigoModulo>1</codigoElemento>
     ...
     <Programa>
       <codigoPrograma>3</codigoElemento>
       ...
       <ElementoRecuperado>
         <codigoElemento>16</codigoElemento>
         ...
         <Tabela>
           <codigoElemento>12</codigoElemento>
           ...
           <Atributo>
             <codigoElemento>5</codigoElemento>
             ...
           </Atributo>
           ...
         </Tabela>
         ...
         <Interface>
           <codigoElemento>16</codigoElemento>
           ...
           <AtributoInterface>
             <codigoElemento>67</codigoElemento>
             ...
           </AtributoInterface>
           ...
         </Interface>
```

Figura 65 – Descrições XML do planejamento da estratégia de migração da etapa "Migração de Pedidos para Web"

4.5 Migrar Aplicação Legada

Nesta fase faz-se a migração da aplicação legada na ferramenta Apyon Studio. Neste estudo de caso a migração foi realizada pelos Participantes 3 e 4.

Primeiramente, importaram-se as descrições XML do planejamento da estratégia de migração da etapa a ser migrada, obtidas na fase anterior. Ao importar o arquivo XML contendo as informações sobre o modelo de negócio da aplicação e os requisitos do planejamento da migração da etapa no Apyon Studio, podem-se reespecificar as interfaces da aplicação, adicionando novas interfaces e componentes para a execução da aplicação, ou movendo ou alterando as interfaces e componentes já existentes.

Na Figura 66 são apresentadas as informações importadas no Apyon Studio, referentes à etapa "Migração de Pedidos para *Web*", cujo objetivo é a disponibilização de suas funcionalidades na Web. No Apyon Studio, as interfaces identificadas podem ser visualizadas no item "Windows" (1) da *treeview* da janela "User Interface Manager". Conforme mostra a janela "Folder – Fields Properties" (2), a interface e os objetos de interface do programa "ven030" foram re-especificados. Foram alterados os nomes e *labels* da interface e de alguns objetos de interface.

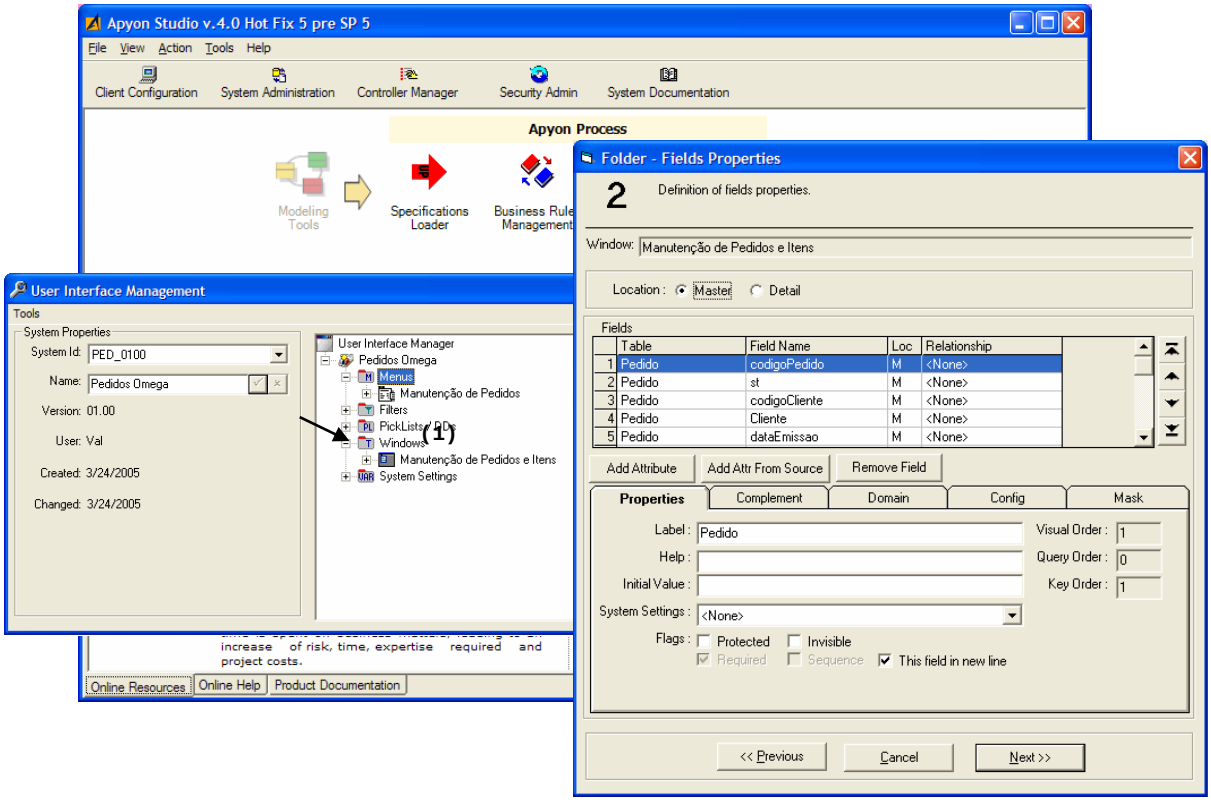


Figura 66 – Descrições XML da etapa "Migração de Pedidos para Web" importadas no Apyon Studio

Em seguida, através do gerador de código, Application Generator, do Apyon Studio, foi gerado o código da interface Pedidos para *Web*, conforme é mostrado na Figura 67.

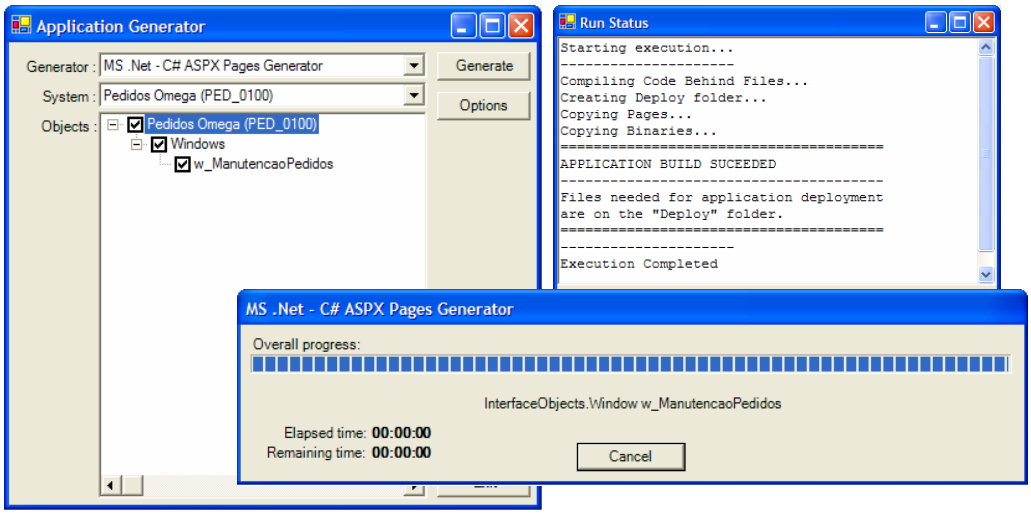


Figura 67 – Geração da interface Pedidos em ASPX na ferramenta Apyon Studio

Na Figura 68 tem-se a interface legada do programa "ven030" antes da aplicação da abordagem AMGraA.

Figura 68 – Interface Legada "ven030"

Na Figura 69 é apresentada a nova interface para a manutenção de Pedidos para Web, gerada pelo Apyon Studio, referente à etapa "Migração de Pedidos para *Web*".

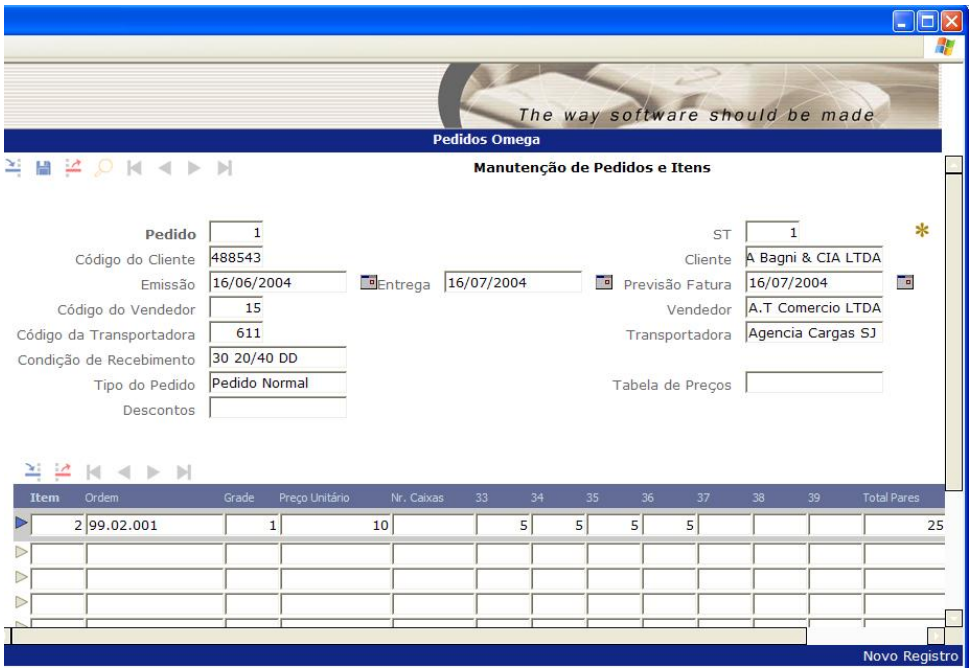


Figura 69 - Nova interface Pedidos em ASPX

Para que o acesso à aplicação legada ainda não migrada através da interface Web gerada, Manutenção de Pedidos e Itens ficasse transparente para o usuário foi construído um *Web Service*, conforme o planejamento da etapa "Disponibilização de serviços através de *Web Service*".

O *Web Service* foi construído na linguagem C# e possui métodos que manipulam as informações legadas e podem ser acessadas por qualquer aplicação.

Ao longo do processo de modernização, os programas "ven010" e "ven015" foram alterados para receber parâmetros da nova aplicação e funcionar em modo "batch", conforme o planejamento da etapa "Alteração de Clientes e Vendedores para possibilitar seu funcionamento em modo "batch"". Nesta etapa a aplicação legada foi alterada, mas apenas para receber parâmetros, a aplicação continua a existir e os usuários continuam usando a aplicação exatamente da mesma forma que usavam antes. Ou seja, para os usuários nada está sendo alterado. A alteração foi realizada para adicionar recursos para receber parâmetros quando a funcionalidade for chamada a partir de outro lugar. Essas modificações refletem na nova aplicação, uma vez que a nova aplicação chama o código legado e este permite o recebimento de parâmetros de outra aplicação. Assim, a nova interface Web acessa as regras de negócio de Clientes e Vendedores em COBOL através da passagem de parâmetros, ou seja, em modo "batch".

De acordo com a Figura 70, a nova interface de Pedidos recupera dados de Clientes e Vendedores ao efetuar um novo pedido, chamando os métodos "RecuperaDadosCli" (1) e "RecuperaDadosVen" (2) do *Web Service*, passando como parâmetros o código de cliente e vendedor, respectivamente. O *Web Service* executa os métodos (1.1) e (2.1) e retorna os resultados em arquivos XMLs, que são tratados pela aplicação. Desta forma, o usuário sempre estará lidando com o *Web Service*, ao invés de lidar com o programa COBOL, que está sendo executado por trás a partir de uma chamada externa.

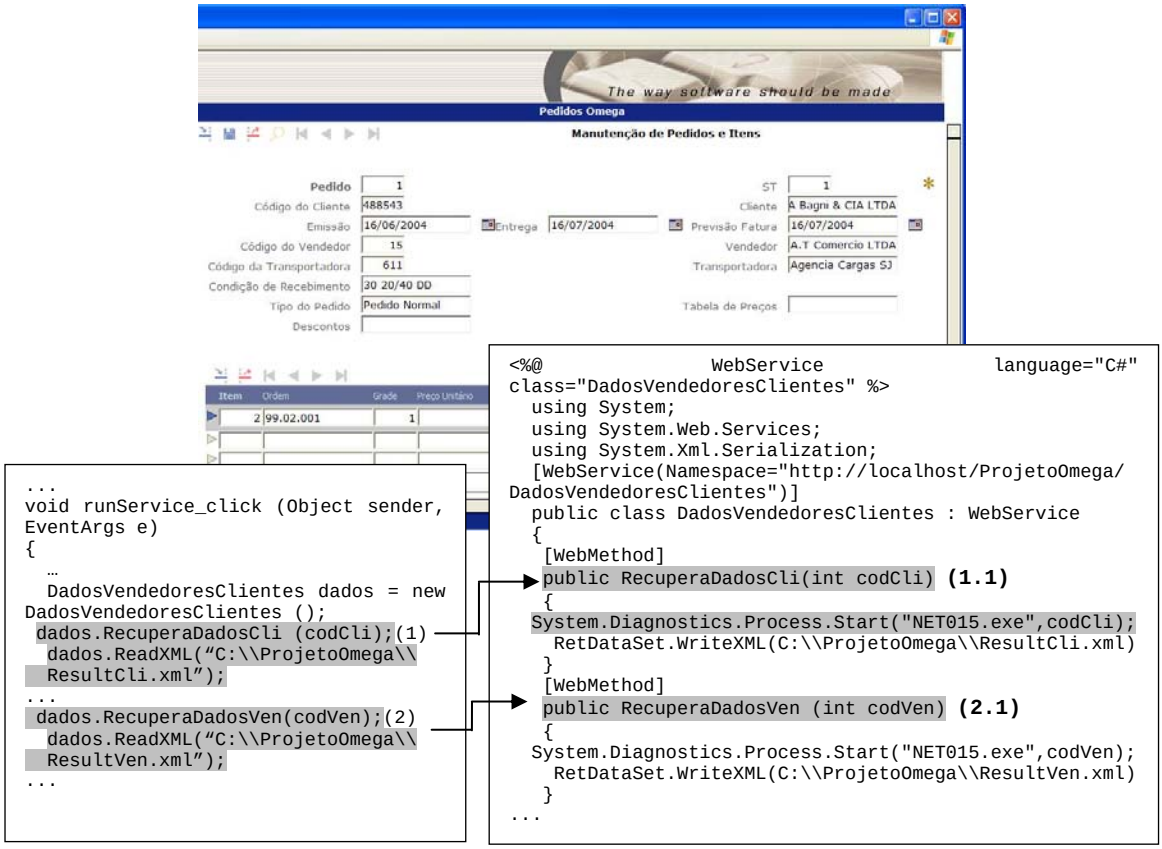


Figura 70 - Nova interface Pedidos e a chamada do serviço do Web Service

O serviço chamado retorna um arquivo XML como resultado. Na Figura 71 é mostrado o arquivo XML resultante da execução do serviço RecuperaDadoVen() para o vendedor de código 002, apresentando as informações do Vendedor de código 002.

```
<VEND>
<CODIGO> 002  </CODIGO>
<RAZSOC> LARANJA COM E REPRES LTDA </RAZSOC>
<ENDERECO> AV JOAO CERNACH,1400 AP-11 BL-06 1.AND.</ENDERECO>
<BAIRRO> CENTRO </BAIRRO>
<CIDADE> BIRIGUI </CIDADE>
<ESTADO> SP </ESTADO>
<CEP> 16200000 </CEP>
<DDD1> 018 </DDD1>
<FONE1> 6426935 </FONE1>
<STATUS> 1 </STATUS>
</VEND>
```

Figura 71 – Arquivo XML com resultado da execução do serviço RecuperaDadoVen()

Em seguida, os programas "ven010" e "ven015" foram também modificados para utilizar um banco de dados e não mais arquivos para armazenamento de seus dados, conforme o planejamento das etapas "Migração de Clientes" e "Migração de Vendedores".

Para a migração dos arquivos de dados para um banco de dados, foi escolhido o SGBD SQL Server 2000. O banco de dados foi construído baseado nas FDs, que contém a estrutura dos arquivos de dados do COBOL.

Na Figura 72 é mostrada a leitura da FD ARQCLI, que possui informações sobre clientes, e a inserção dos dados no banco de dados através de comandos SQL. Como alguns campos da base de dados estavam com caracteres inválidos, possivelmente por não terem sido inicializados na gravação, foi necessário tratar tais valores. No COBOL, quando isso acontece, é atribuído nulo para o campo. Portanto, foi necessário tratar estes campos, pois o valor nulo para o COBOL difere do valor NULO do banco de dados. Em (1) é feito o tratamento para verificar se o valor do campo é nulo e em (2) os valores são inseridos no banco de dados na tabela Clientes.

```
ENVIRONMENT     DIVISION.
  DATA            DIVISION.
  WORKING-STORAGE SECTION.
  PROCEDURE       DIVISION.

ROTINA.
     ADD 1 TO WS-LIDOS.
     DISPLAY  WS-LIDOS AT 1033
     MOVE CLI-RAZSOC         TO TCLIENTES-RAZSOC.
     MOVE CLI-CODIGO         TO TCLIENTES-CODIGO.
     IF CLI-CGCCPF = ZEROS OR                          (1)
        CLI-CGCCPF NOT NUMERIC
        MOVE -1              TO TCLIENTES-CGCCPF-N
     ELSE
        MOVE ZEROS           TO TCLIENTES-CGCCPF-N
        MOVE CLI-CGCCPF      TO TCLIENTES-CGCCPF
     END-IF
     ...
     EXEC SQL
          SELECT MAX(CODIGO) FROM OMEGA.DBO.CLIENTES INTO :WS-CODIGO-SQL
    END-EXEC
     ADD 1 TO WS-CODIGO-SQL
     MOVE WS-CODIGO-SQL TO TCLIENTES-CODIGO
    EXEC SQL
              INSERT INTO OMEGA.DBO.CLIENTES           (2)
                     (
                      CODIGO,
                      RAZSOC,
                      CGCCPF,
                      RGINSCEST,
                       ...
                      )
                     VALUES
                     (:TCLIENTES-CODIGO,
                      :TCLIENTES-RAZSOC,
                      :TCLIENTES-CGCCPF:TCLIENTES-CGCCPF-N,
                      :TCLIENTES-RGINSCEST:TCLIENTES-RGINSCEST-N,
                      :TCLIENTES-PESSOA:TCLIENTES-PESSOA-N,
```

Figura 72 – Acesso a Banco de Dados de Clientes

Ao longo do processo de modernização da aplicação Ômega, outras etapas foram definidas, como as que objetivavam a migração das interfaces da aplicação para a linguagem C#. Na Figura 73 tem-se a nova execução da aplicação Ômega após a aplicação da abordagem AMGraA.

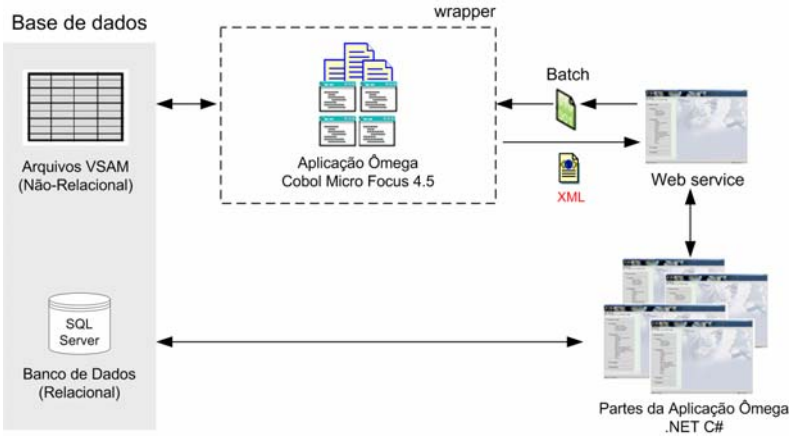


Figura 73 - Novo modelo de execução da aplicação Ômega

Os resultados para o projeto Ômega foram obtidos em três semanas de aplicação da abordagem, sem considerar o tempo gasto na escrita das transformações de *software* para a fase de análise e extração de informações do modelo de negócio.

É importante citar que no Apyon Studio os códigos são gerados e compilados, permitindo que a aplicação funcione de forma independente do Apyon Studio.

4.6 Validar Migração da Aplicação

A fase de validação da migração visa projetar casos de testes específicos para os cenários abordados na migração da aplicação legada.

Os testes realizados neste projeto são os que seguem a abordagem chamada teste caixa-preta, onde os testes realizados demonstram que as funções dos módulos e a integração estão operacionais, através de uma entrada aceita e uma saída correta produzida.

Para a validação da Etapa de Migração de Pedidos para Web, o Engenheiro de *Software* definiu os seguintes casos de teste:

- Verificar limite de crédito: cadastrar um pedido para um cliente "A Bagni & CIA LTDA" e verificar se o limite de crédito é válido para este cliente;
- Cadastrar pedido: inserir um pedido já cadastrado para o cliente "A Bagni & CIA LTDA" emitido em "16/06/2004", entregue em "16/07/2004 e com faturamento para "16/07/2004", verificando se a mensagem retornada é "Pedido já cadastrado!";
- Verificar vendedor: para o pedido de número "100" verificar se o vendedor responsável é "SERGIO LUIZ CANCIAN";

Os resultados dos casos de testes aplicados podem ser obtidos da aplicação legada, pois devem ter os mesmos resultados, salvo onde houve modificação, por exemplo, alteração da mensagem informada para o usuário.

O projeto de casos de teste é uma atividade ampla e desafiante e, portanto, não foi aprofundada neste projeto de pesquisa. Esta fase será melhor abordada em trabalhos futuros na Fase II do Projeto Fapesp.

Capítulo 5

Conclusões

Com objetivo de elaborar uma abordagem para apoiar a modernização de aplicações legadas nas empresas, um estudo das abordagens de modernização existentes na literatura foi realizado, levantando as diversas técnicas utilizadas ao longo dos anos.

Considerando a necessidade das empresas em modernizar suas aplicações, de acordo com suas prioridades, que muitas vezes são influenciadas por fatores externos, exigindo flexibilidade e rapidez na obtenção dos resultados; a necessidade em preservar a importância da aplicação legada; a familiaridade que os mantenedores e usuários têm com a aplicação; e a execução das funcionalidades existentes durante o processo de modernização, verificou-se a necessidade de realizar a modernização das aplicações de forma gradativa.

Diante dessas necessidades, alguns requisitos para a migração gradativa foram identificados e discutidos no Capítulo 2. Considerando os requisitos identificados, o GOES (Grupo de Engenharia de *Software*), a Apyon Technology, empresa atuante no desenvolvimento de *software*, juntamente com o apoio da FAPESP para o desenvolvimento de pesquisa nas pequenas empresas, pesquisaram uma abordagem para migração gradativa de aplicações legadas denominada AMGraA. Para validação da abordagem proposta foi realizado um estudo de caso, apresentado no Capítulo 4.

A seção 5.1 apresenta uma síntese dos principais resultados obtidos com a realização deste trabalho de pesquisa. A seção 5.2 apresenta alguns trabalhos correlatos. A seção 5.3 apresenta uma análise crítica sobre o trabalho realizado. A seção 5.4 apresenta os artigos publicados. A seção 5.5 sugere alguns trabalhos futuros.

5.1 Síntese dos Principais Resultados

Os principais resultados obtidos com o desenvolvimento deste trabalho foram:

- Discussão de algumas abordagens para modernização de aplicações legadas existentes na literatura;
- Identificação de alguns requisitos necessários para uma abordagem de migração gradativa ideal;
- Elaboração da abordagem gradativa para modernização de aplicações legadas, AMGraA, que atende os requisitos identificados neste trabalho;
- Construção do domínio COBOL e do transformador COBOL no ST Draco-PUC para extração dos dados do modelo de negócio da aplicação;
- Construção do transformador para transformação dos fatos coletados pelo ST Draco-PUC em descrições no formato XML;
- Construção da ferramenta Migration Project Planning (MPP) para apoiar Engenheiro de *Software* no planejamento do projeto de migração na abordagem AMGraA; e
- Integração do ST Draco-PUC a um produto de mercado, Apyon Studio, para oferecer um ambiente inovador e muito solicitado por grandes empresas que desejam migrar suas aplicações legadas para novas tecnologias.

5.2 Trabalhos Correlatos

Embora existam várias abordagens para a migração de aplicações na literatura, ainda há uma carência por métodos que sejam viáveis para a aplicação na indústria, que ofereçam resultados práticos e que permitam o aproveitamento de investimentos anteriores em projetos de modernização realizados pela empresa.

A maior parte das abordagens trata somente a mudança para o paradigma orientado a objetos, como em [JACOBSON; LINDSTROM 1991], [MARKOSIAN 1994], [GALL; KLÖSH 1994], [PENTEADO 1996], [SNEED 1996], [RIVA 2000], [FONTANETTE et al. 2002b] e [WILKENING 1995], obtendo alguma representação em um nível mais alto de abstração. Em [GALL; KLÖSH 1994], Gall e Klösh recuperam diagramas de fluxo de dados (DFD's), diagrama de entidade e relacionamento (DER) e um modelo orientado a objetos da aplicação. Já as abordagens mais recentes como em [FONTANETTE et al. 2002b], [ALVARO et al. 2003] recuperam modelos de casos de uso e seqüência da aplicação.

Outra técnica utilizada é o encapsulamento da aplicação como o proposto em [GALL; KLÖSH 1994]. O encapsulamento pode muitas vezes ser tão complexo e exigir um grande número de mudanças nos programas principais e periféricos, que é quase uma reescrita da aplicação. O

tempo e recursos gastos podem ser iguais, já que uma tecnologia nova também acompanha ferramentas novas de desenvolvimento rápido para agilizar o processo.

O método RST [FONTANETTE et al. 2002a], [FONTANETTE et al. 2002b], [FONTANETTE et al. 2002c], [FONTANETTE et al. 2002d], [FONTANETTE et al. 2002e], [JESUS 2000], [FUKUDA 2000], [NOVAIS 2002] propõe uma abordagem para a conversão de aplicações através de transformações de *software*. O processo é semi-automatizado por transformadores. Contudo, a escrita destes transformadores exige alto grau de conhecimento das sintaxes e semânticas das linguagens de origem e de destino. Ao recuperar o projeto orientado a objetos, o Engenheiro de *Software* utiliza uma ferramenta CASE para reespecificar a aplicação para realizar refinamentos e re-projetar a aplicação em uma nova arquitetura, por exemplo, orientada a componentes. A implementação do código pode ser realizada através da ferramenta CASE ou através dos transformadores de geração de código do ST Draco-PUC.

O processo proposto pelo método RST é um processo de migração não-gradativa. Este processo não trata a conversão por partes de uma aplicação, mantendo a integração com o restante da aplicação legada, como acontece quando a empresa deseja converter somente um de seus módulos para Web, por exemplo. Não há uma flexibilidade de planejar a migração, de estudar o que poderia ser migrado primeiro para, por exemplo, atender os orçamentos das empresas. A flexibilidade é muito importante, visto que hoje a integração entre suas aplicações é um grande problema que as empresas estão tentando solucionar.

O processo proposto por Wilkening em [WILKENING et al. 1995] pode ser tão complexo e exigir muita mão-de-obra especializada para entender e documentar a aplicação que é quase uma reescrita da aplicação. E muitas vezes não adianta apenas entender o que existe, necessitando também entender o negócio, que também pode ter mudado, pois existe um backlog enorme de solicitações de usuários e existem vários erros conhecidos. Outros fatores como mudanças de decisões da empresa ou lançamento de produtos diferentes, sem contar a globalização e mudanças na economia do país, que freqüentemente ocorrem, exigem mudanças nas aplicações.

Em [BISBAL; LAWLESS 2003] os autores falam de 3 tipos de migração - "Cut-and-run", onde é realizada uma parada total da aplicação antiga; "Phased", onde a migração é realizada por partes e "Parallel", onde ambas as aplicações são mantidas. Para os autores, a migração geralmente passa por um *gateway,* onde existe um *middleware* para que a aplicação nova converse com a aplicação antiga, para que a aplicação antiga seja então migrada aos poucos. Esta é justamente a forma mais cara de se fazer e a mais onerosa, pois uma aplicação que vai ser executada como um *middleware* provavelmente terá que ser escrita novamente logo em seguida porque vai ficar

"presa" ao *middleware*. Em alguns casos eles citam *gateway* apenas como uma infra-estrutura para permitir que as aplicações conversem, mas a programação tem que ser feita totalmente à mão. No processo definido pela AMGraA, as 3 opções podem ser usadas. Ainda melhor, cada uma destas 3 opções pode ser escolhida de acordo com o módulo a ser migrado.

5.3 Análise Crítica e Limitações

Os seguintes pontos deste trabalho de pesquisa podem ser destacados:

- **Recuperação de dados**. Inicialmente foi proposta a recuperação dos trechos de código legado em que um elemento era recuperado e a sua visualização na ferramenta MPP. Mas devido ao ST Draco-PUC ler uma árvore sintática e não sequencialmente um arquivo texto, outros estudos serão necessários para realização desta tarefa. Esta atividade é um diferencial entre as abordagens de migração e, portanto, será considerada em trabalhos futuros;
- **Decomposição da aplicação**. A tarefa de decomposição da aplicação em módulos, proposta pela abordagem, não pôde ser refinada, uma vez que a aplicação Ômega utilizada para a validação da abordagem estava particionada em módulos. Portanto, novos estudos de casos devem ser realizados para validar a decomposição da aplicação em módulos;
- **Biblioteca de transformação**. A biblioteca de transformação DracoKBXML construída no ST Draco-PUC para transformar os fatos coletados pelo ST Draco-PUC em formato XML, é independente de linguagem e, portanto, poderá ser reutilizada em qualquer domínio;
- **Separação da tecnologia de negócio**. A abordagem AMGraA baseia-se na separação do negócio da aplicação da parte tecnológica (MDA). Esta separação permite uma migração gradativa das aplicações, reduzindo a complexidade envolvida nos processos de modernização de aplicações legadas e também no tempo, uma vez que os resultados são acompanhados gradativamente, conforme a migração da aplicação;
- **Integração das ferramentas**. As ferramentas ST Draco-PUC, Migration Project Planning (MPP) e Apyon Studio foram integradas para facilitar o trabalho do Engenheiro de *Software*, atendendo ao requisito Integração entre ferramentas, identificado no Capítulo2 . O ST Draco-PUC e o Apyon Studio foram desenvolvidos isoladamente, mas foram complementares em suas funções no caso de migração de aplicações legadas. Com a integração dessas três ferramentas, aliada a uma abordagem de migração gradativa, tem-se um ambiente para conversão e modernização de aplicações legadas;

É importante considerar que as ferramentas ST Draco-PUC e Apyon Studio foram desenvolvidas a partir de trabalhos acadêmicos, e que o Apyon Studio está disponível no

mercado há mais de seis anos, enquanto que o ST Draco-PUC recentemente vem sendo utilizado na indústria. A abordagem foi definida objetivando aproveitar os melhores recursos de cada uma dessas ferramentas, com foco em dois objetivos comuns: a migração de aplicações legadas e uma solução mais amigável de reconstrução e modernização de *software* para as empresas. Desta forma, estas podem estudar a melhor forma de migrar suas aplicações em conformidade com seus objetivos e prioridades, de modo que esta tenha um baixo impacto, alto nível de produtividade, alta qualidade e baixos custos.

- Comunicação entre módulos. A abordagem proposta oferece a comunicação entre os módulos migrados e legados através de algumas tecnologias discutidas anteriormente, atendendo ao requisito Comunicação entre módulos, identificado no Capítulo 2. Mas, o Apyon Studio ainda não oferece a geração automática de *Web Services*, por exemplo.

5.4 Publicações

Os seguintes artigos foram publicados como resultado deste trabalho de pesquisa:

- FONTANETTE, V. ; PRADO, Antonio Francisco Do ; OLIVEIRA, André Luis Costa de . Uma Abordagem para Migração Gradativa de Aplicações Legadas. In: IX Workshop de Teses e Dissertações em Engenharia de *Software* - XVIII Simpósio Brasileiro de Engenharia de *Software*, 2004, Brasília. IX Workshop de Teses e Dissertações em Engenharia de *Software* - XVIII Simpósio Brasileiro de Engenharia de *Software*, 2004.

- MORAES, João Luis Cardoso de ; BOSSONARO, Adriano Aleixo ; FONTANETTE, V. ; LUCRÉDIO, Daniel ; GARCIA, Vinicius Cardoso ; PRADO, Antonio Francisco Do . An Approach for Construction and Reuse of *Software* Components Frameworks implemented in Delphi. In: International Symposium on Advanced Distributed Systems (ISADS'2004), 2004, Guadalajara Jalisco. The Fourth International Symposium on Advanced Distributed Systems (ISADS'2004), 2004.

- PERES, Darley Rosa ; ALVARO, Alexandre ; FONTANETTE, V. ; GARCIA, Vinicius Cardoso ; PRADO, Antonio Francisco Do ; BRAGA, Rosana Teresinha Vaccare . TB-REPP - Padrões de Processo para Engenharia Reversa baseado em Transformações. In: The Third Latin American Conference on Pattern Languages of Programming (SugarLoafPlop), 2003, Porto de Galinhas - Pernambuco. The Third Latin American Conference on Pattern Languages of Programming (SugarLoafPlop), 2003.

- BOSSONARO, Adriano Aleixo ; MORAES, João Luis Cardoso de ; FONTANETTE, V. ; GARCIA, Vinicius Cardoso ; PRADO, Antonio Francisco Do . Implementações de Frameworks de Componentes, dirigidas por Modelos do Método Catalysis. In: The Fourth Congress of Logic

Applied to Technology (LAPTEC'2003), 2003, Marília. Proceedings of The Fourth Congress of Logic Applied to Technology (LAPTEC'2003),, 2003.

- FONTANETTE, V. ; GARCIA, Vinicius Cardoso; PERES, A. B. ; PRADO, Antonio Francisco Do ; SANT'ANNA, M. H. B. . Component-Oriented *Software* Reengineering using Transformations. In: International Conference on Computer Science, *Software* Engineering, Information Technology, e-Business, and Applications, 2002, Foz do Iguaçu.

- FONTANETTE, V. ; GARCIA, Vinicius Cardoso ; PERES, A. B. ; PRADO, Antonio Francisco Do ; SANT'ANNA, M. H. B. . Component-Oriented *Software* Reengineering using Transformations. In: International Conference on Computer Science, *Software* Engineering, Information Technology, e-Business, and Applications, 2002, Foz do Iguaçu.

- FONTANETTE, V. ; GARCIA, Vinicius Cardoso ; PERES, A. B. ; BOSSONARO, Adriano Aleixo ; PRADO, Antonio Francisco Do . Reprojeto de Sistemas Legados Baseado em Componentes de *Software*. In: XXVIII Conferencia Latinoamericana de Informática (InfoUYclei), 2002, Montevideo - Uruguai. XXVIII Conferencia Latinoamericana de Informática (InfoUYclei). Montevideo : Mastergraf SRL, 2002. v. 1. p. 177-177.

- FONTANETTE, V. ; GARCIA, Vinicius Cardoso ; PERES, A. B. ; BOSSONARO, Adriano Aleixo ; PRADO, Antonio Francisco Do . Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações. In: II Workshop Chileno de Ingeniería de *Software*, 2002, Copiapó - Chile. II Workshop Chileno de Ingeniería de *Software*, 2002. p. 1-10.

- FONTANETTE, V. ; GARCIA, Vinicius Cardoso ; PERES, A. B. ; BOSSONARO, Adriano Aleixo ; PRADO, Antonio Francisco Do . Reengenharia de *Software* usando Transformações (RST). In: The Second Ibero-American Symposium on *Software* Engineering and Knowledge Engineering (JIISIC/2002), 2002, Salvador - Bahia. The Second Ibero-American Symposium on *Software* Engineering and Knowledge Engineering (JIISIC/2002), 2002.

- GARCIA, Vinicius Cardoso ; FONTANETTE, V. ; PERES, A. B. ; BOSSONARO, Adriano Aleixo ; PRADO, Antonio Francisco Do . DDE - Draco Domain Editor. In: XVI SIMPÓSIO BRASILEIRO DE ENGENHARIA DE *SOFTWARE*, 2002, Gramado-RS. XVI Simpósio Brasileiro de Engenharia de *Software*, 2002. v. 1. p. 378-383.

- FONTANETTE, V. ; GARCIA, Vinicius Cardoso ; PERES, A. B. ; BOSSONARO, Adriano Aleixo ; PRADO, Antonio Francisco Do . Estratégia de Reengenharia de *Software* Baseada em Componentes Distribuídos. In: II WORKSHOP DE DESENVOLVIMENTO BASEADO EM COMPONENTES (WDBC 2002), 2002, Itaipava - RJ. II WORKSHOP DE DESENVOLVIMENTO BASEADO EM COMPONENTES, 2002.

5.5 Trabalhos Futuros

No decorrer do desenvolvimento deste trabalho, observaram-se, dentre outras, as seguintes oportunidades para trabalhos futuros, citadas a seguir:

- Refinamento e complementação do domínio e do transformador COBOL com novos padrões de reconhecimento para atender às variações da linguagem COBOL de diferentes fabricantes;
- Aperfeiçoamento da ferramenta MPP para permitir a visualização dos trechos de código que deram origem à identificação dos dados do modelo de negócio, como tabelas, interfaces e regras de negócio;
- Desenvolvimento de um mecanismo de controle de versões para a importação das informações do modelo de negócio na ferramenta MPP;
- Geração automática da camada de *Web Services* pela ferramenta Apyon Studio, visando facilitar as tarefas do Engenheiro de *Software* ao desenvolver a forma de comunicação e integração entre módulos e demais aplicações;
- Recuperação de modelos em alto nível de abstração para facilitar o entendimento da aplicação legada e facilitar a manutenção;
- Os requisitos funcionais da aplicação, como as regras de negócio, são convertidos em uma nova linguagem com apoio do ST Draco-PUC, ou se o Engenheiro de *Software* preferir, manualmente. Os conceitos de *Refactoring* [ROBERTS 1999] podem ser usados na conversão das regras de negócio para melhorar a estrutura do código, eliminar trechos de código "morto", facilitando o seu entendimento e a sua manutenção. Através do uso das técnicas de *Refactoring* é possível eliminar a aparência da estrutura da linguagem legada deixada pela conversão entre códigos.
- Aperfeiçoamento das atividades de testes para validação e implantação das partes migradas;
- Submissão do Projeto da Fase II do Projeto Fapesp PIPE, visando:
  - Refinamentos na abordagem para corrigir possíveis erros;
  - Estudo de novas tecnologias;
  - Aprimoramento das ferramentas MPP e Apyon Studio;
  - Experimentos;
  - A divulgação da abordagem AMGraA nas empresas;

- Desenvolvimento de aplicações "exemplos", que implementam os diferentes cenários vivenciados atualmente pelas empresas, com possíveis soluções de modernização, integração e compartilhamento de dados;
- Transferência da abordagem para as empresas com demonstrações, utilizando essas aplicações "exemplos", para apresentar o conceito estudado e desenvolvido para facilitar o entendimento e a aceitação das empresas e ainda fazer refinamentos na abordagem.

Capítulo 6

Referência Bibliográfica

[ALVARO et al. 2003] ALVARO, A. et al. Orion-RE: A Component-Based Software Reengineering Environment. In: Proceedings of the 10th Working Conference on Reverse Engineering (WCRE). [S.l.]: IEEE Computer Society Press, 2003.

[APYON 2000] APYON TECHNOLOGY. Documentação de especificação de diagramas ER em Ferramentas CASE. São Paulo: Apyon Technology ltda, 2000.

[BIANCHI et al. 2003] BIANCHI, A.; CAIVANO, D.; VISAGGIO, G. Iterative Reengineering of Legacy Systems. IEEE Transactions on Software Engineering v.29, n.3, p. 225-241, March, 2003.

[BIGGERSTAFF et al. 1994] BIGGERSTAFF, T. J.; MITBANDER, B. G.; WEBSTER, D. E. Program understanding and the concept assignment problem. Communications of the ACM, ACM Press, v. 37, n. 5, p. 72–82, 1994. ISSN 0001-0782.

[BISBAL; LAWLESS 2003] BISBAL, J., LAWLESS, D., WU, B., GRIMSON, J.. Legacy Information Systems: Issues and Directions, IEEE Software, September/October 1999.

[CHIKOFSKY; CROSS 1990] CHIKOFSKY, E. J., CROSS, J. H. Reverse Engineering and Design Recovery: a Taxonomy. IEEE Software, p. 13-17, janeiro 1990.

[COYLE 2000] COYLE, F. Legacy Integration Changing Perspectives. IEEE Software, March/April 2000.

[ELMASRI; NAVATHE 2000] ELMASRI, R., NAVATHE, S. B. "Fundamentals of Database Systems", Addison-Wesley, 3rd Edition, 2000.

[FONTANETTE 2004] FONTANETTE, V. Uma Abordagem para Migração Gradativa de Aplicações Legadas. Monografia de Qualificação. Departamento de Computação, UFSCar, 2004.

[FONTANETTE et al. 2001] FONTANETTE, V. et al. RHAE/CNPQ - Projeto: Reengenharia de Software Usando Transformações (RST). 2001. Processo número: 610.069/01-2.

[FONTANETTE et al. 2002a] FONTANETTE V. et al. Component-Oriented Software Reengineering using Transformations. International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications - (CSITeA'02). Pág. 206 - 211. ISBN: 0-9700776-3-7. Foz do Iguaçu, Brasil. 6-8 de Junho de 2002, 2002a.

[FONTANETTE et al. 2002b] FONTANETTE V. et al. Component-Oriented Software Reengineering using Transformations. International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications - (CSITeA'02). Pág. 206

- 211. ISBN: 0-9700776-3-7. Foz do Iguaçu, Brasil. 6-8 de Junho de 2002, 2002b.

[FONTANETTE et al. 2002c] FONTANETTE, V. et al. Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações. II Workshop Chileno de Ingeniería de Software (JCC'2002), Copiapó-Chile, 2002c.

[FONTANETTE et al. 2002d] FONTANETTE, V.; GARCIA, V. et al. Reengenharia de Software usando Transformações (RST). In: The Second Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC/2002) - Sessão Técnica(4), Artigo nº3, 2002, Salvador - Brazil. (Documentação em CD-ROM), 2002d.

[FONTANETTE et al. 2002e] FONTANETTE, V. et al. Reprojeto de Sistemas Legados Baseado em Componentes de Software. In: XXVIII Conferencia Latinoamericana de Informática (InfoUYclei), nov., 2002, Montevideo, Uruguai. Anais. Montevideo: Mastergraf SRL, 2002. 177p. CL89. ISBN: 9974-7704-1-6 (Documentação em CD-ROM), 2002e.

[FUKUDA 2000] FUKUDA, A. P. Refinamento Automático de Sistemas Orientados a Objetos Distribuídos, 2000 - Dissertação de Mestrado, Ciências da Computação, UFSCAR – Universidade Federal de São Carlos.

[GALL, KLÖSH 1993] GALL, H.; KLÖSCH, R. Capsule oriented reverse engineering for software reuse. European Software Engineering Conference (ESEC'93), Lecture Notes in Computer Science(LNCSD9), v. 717, p. 418–433, 1993. ISSN 0302-9743.

[GALL, KLÖSH 1994] GALL, H.; KLÖSCH, R. Program transformation to enhance the reuse potential of procedural software. In: Proceeding of the ACM Symposium on Applied Computing (SAC'1994). [S.l.]: ACM Press, 1994. p. 99–104. ISBN 0-89791-647-6.

[HERZUN 2002] HERZUN, P. Web Services and Service-Oriented Architectures. Executive report, vol. 4, no. 10. Cutter Distributed Enterprise Architecture Advisory Service, 2002.

[JACOBSON; LINDSTROM 1991] JACOBSON, I., LINDSTROM, F. Re-engineering of old systems to an object-oriented architecture. Conference on Object Oriented Programming Systems, Languages and Applications (OOPSLA).Proceedings. 1991, p. 340-350.

[JESUS 2000] JESUS, E. S. Engenharia Reversa de Sistemas Legados Usando Transformações, 2000 - Dissertação de Mestrado, Ciências da Computação, UFSCAR - Universidade Federal de São Carlos.

[LEITE et al. 1994] LEITE, J. C. S. P., SANT'ANNA, M., FREITAS, F. Draco-PUC: A Technology Assembly for Domain Oriented Software Development. In: Proceedings of the 3rd International Conference on Software Reuse (ICSR'94). [S.l.]: IEEE Computer Society Press, 1994. p. 94–100.

[LEITE et al. 1996] LEITE, J. C. S. P., SANT'ANNA, M., PRADO, A. F. Porting Cobol Programs using Transformational Approach. Journal of Software Maintenance: Research and Practice, John Wiley & Sons Ltd, v. 9, p. 3–31, October 1996.

[MARKOSIAN et al. 1994] MARKOSIAN, L. et al. Using an enabling technology to reengineer legacy systems. Communications of the ACM, ACM Press, v. 37, n. 5, p. 58–70, 1994. ISSN 0001-0782.

[MARTIN; MCCLURE 1983] MARTIN, J., MCCLURE, C. Software Maintenance: The Problems and Its Solutions. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[MIT 2003] MIT, Softex e W-Class. A indústria do software no Brasil – 2002. Fortalecendo a economia do conhecimento. Campinas : SOFTEX, 2002. 80 p.

[NEIGHBORS 1984] NEIGHBORS, J.M. The Draco approach to Constructing Software from

Reusable Components. IEEE Transactions on Software Engineering v.se-10, n.5, pp.564-574, September, 1984.

[NOGUEIRA 2002] NOGUEIRA, A R. Transformação de DataFlex Procedural para Visual DataFlex Orientado a Objetos reusando um Framework, 2002 - Dissertação de Mestrado, Ciências da Computação, UFSCAR - Universidade Federal de São Carlos, Antonio Francisco do Prado.

[NOVAIS 2002] NOVAIS, E. R. A. Reengenharia de Software Orientada a Componentes Distribuídos, 2002 - Dissertação de Mestrado, Ciências da Computação - UFSCAR - Universidade Federal de São Carlos, Antonio Francisco do Prado

[OLIVEIRA 1998] OLIVEIRA, A. L. C. Metodologia para desenvolvimento de Sistemas de Informação através da utilização de módulos autônomos. Dissertação de Mestrado. UFRGS, Porto Alegre, 1998.

[OLIVEIRA 2004] OLIVEIRA, A. L. C.. Uma Abordagem para Migração Gradativa de Aplicações Legadas. FAPESP/PIPE - Processo: 03/07851-4. Data inicio: 01/05/04.

[OLIVEIRA; PALAZZO 1999] OLIVEIRA, A. L. C.; PALAZZO, J. M. O. Uma arquitetura para reduzir a complexidade e aumentar a produtividade do Ciclo de Vida do Desenvolvimento de Sistemas. Simpósio Brasileiro de Engenharia de Software (SBES). Florianópolis, 1999.

[OLSEM 1998] OLSEM, M. R. An incremental approach to software systems re-engineering. Journal of Software Maintenance, John Wiley & Sons, Inc., v. 10, n. 3, p. 181–202, 1998. ISSN 1040-550X.

[PENTEADO 1996] PENTEADO, R.D. Um Método para Engenharia Reversa Orientada a Objetos. São Carlos-SP, 1996. Tese de Doutorado. Universidade de São Paulo. 251p.

[PRESSMAN 1995] PRESSMAN, R. S. Engenharia de Software. Makron Books: São Paulo, 1995

[PRICE et al. 1993] PRICE, B.; BAECKER, R.; SMALL, I. A principled taxonomy of software visualization. Journal of Visual Languages and Computing, ACM Press, v. 4, n. 3, p. 211–266, 1993.

[RIVA 2000] RIVA, C. Reverse architecting: an industrial experience report. In: Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'2000). [S.l.]: IEEE Computer Society Press, 2000. p. 42–50.

[ROBERTS 1999] ROBERTS, D. B. Practical Analysis for Refactoring. Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 1999

[SEACORD et al. 2003] SEACORD, R., PLAKOSH D., LEWIS, A. G.. Modernizing Legacy Systems – Software Technologies, Engineering Processes, and Business Practices. SEI-Series in Software Engineering – Addison-Wesley, 2003. ISBN 0-321-11884-7.

[SNEED 1996] SNEED, H. M. Object-oriented cobol recycling. In: Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE'96). [S.l.]: IEEE Computer Society Press, 1996. p. 169–178.

[SOMMERVILLE 2001] SOMMERVILLE, I. Software Engineering. 6th Edition. Boston: Addison-Wesley Higher Education, 2001.

[STANDISH GROUP 2001] THE STANDISH GROUP INTERNATIONAL, INC. Extreme CHAOS, p.1-12, 2001. Disponível em http://standishgroup.com.

[STRINGHINI 2003a] STRINGHINI, D. Definição e Implementação de Módulos de Mapeamento

Objeto-Relacional no Apyon Studio. CNPq/RHAE - Processo: 552290/02-5, Data de início: 01/07/2003, 2003a. (Em andamento).

[STRINGHINI 2003b] STRINGHINI, D. Modelo Gráfico de Dependência entre as Regras de Negócio e o Impacto Físico/Financeiro sobre a Manutenção. FAPESP/PIPE - Processo: 01/13400-0 (Fase I terminada, aguardando liberação da Fase II) – Fase 1 submetida em 30/11/2001. Fase II submetida em 01/12/2003, 2003b.

[SYSTA 1999] SYSTA, T. The relationships between static and dynamic models in reverse engineering java software. In: Proceedings of the 6th Working Conference on Reverse Engineering (WCRE'99). [S.l.]: IEEE Computer Society Press, 1999.

[TOPLEY 2003] TOPLEY, K. Java Web Services in a Nutshell – A Desktop Quick Reference. O'reilly, ISBN: 0-596-00399-4, Junho 2003.

[ULRICH 1990] ULRICH, W. The Evolutionary Growth of Software Engineering and the Decade Ahead. American Programmer 3, 10: 12-20. 1990.

[WEIDERMAN et al. 1997] WEIDERMAN, N, et al. Implications of Distributes Object Technology for Reengineering, Tech. Report CMU/SEI-97-TR-005, Carnegie Mellon Univ., Pittsburgh, 1997.

[WILKENING et al. 1995] WILKENING, D. E. et al. A reuse approach to software reengineering. Journal of Systems and Software, v. 30, n. 1-2, p. 117–125, 1995. ISSN 0164-1212.

[ZOU; KONTOGIANNIS 2003] ZOU, Y.; KONTOGIANNIS, K. Incremental Transformation of Procedural Systems to Object Oriented Platforms.Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03), IEEE Computer Society Press, 2003.

[CARVALHO 1993] CARVALHO, J. E. M. de. Microsoft COBOL 4.5: Programação Avançada. São Paulo, Makron Books, 1993. p.327

[SAADE 1998] SAADE J.; COBOL Sem Mistério, Editora Novatec, 1998 - ISBN: 85-85184-61-2, p.520.

[SAADE 1997] SAADE J.; COBOL ANS 85 – Guia de Consulta Rápida, Editora Novatec, 1997 - ISBN: 85-85184-57-4, p.32.

Apêndice I

Documentação do Protótipo da Ferramenta Migration Project Planning (MPP)

1 Casos de Uso

A Tabela 1 mostra os principais casos de uso definidos para a ferramenta Migration Project Planning (MPP). Os casos de uso, conforme mostra a tabela serão executados pelo principal ator da aplicação, o Engenheiro de Software.

Tabela 5 – Lista dos Principais Casos de Uso

Nr.	Descrição	Caso de Uso	Eventos	Respostas
1	CadastrarProjeto	Engenheiro de Software cadastra Projeto de Migração de uma aplicação.	dadosProjeto	MSG01
2	DetalharProjeto	Engenheiro de Software detalha projeto	dadosProjeto, dadosDetalhamento	MSG04
3	ImportarModeloNegocio	O Engenheiro de Software importa as informações do modelo de negócio coletadas	dadosModeloNegocio	MSG05
4	ExportarDescricoesEtapa	O Engenheiro de Software exporta as descrições da etapa que pretende migrar	dadosExportacao	MSG06
5	DefinirEtapasDeMigracao	O Engenheiro de Software define as	dadosEtapa	MSG07

		etapas do seu projeto de migração		
6	AnalisarInterface	O Engenheiro de Software analisa dados coletados para interfaces de usuário da aplicação	codigoInterface	dadosInterface dadosAtributos
7	AnalisarTabela	O Engenheiro de Software analisa dados coletados para as tabelas	codigoTabela	dadosTabela dadosCampos

MSG01: Projeto cadastrado com sucesso / Projeto atualizado com sucesso / Projeto excluído com sucesso.

MSG02: Detalhamento cadastrado com sucesso / Detalhamento atualizado com sucesso / Detalhamento excluído com sucesso.

MSG03: Dados importados com sucesso / Dados atualizados com sucesso / Dados excluídos com sucesso.

MSG04: Dados exportados com sucesso / Dados não exportados

MSG05: Etapa cadastrada com sucesso / Etapa atualizada com sucesso / Etapa excluída com sucesso.

MSG06: Interface cadastrada com sucesso / Interface atualizada com sucesso / Interface excluída com sucesso.

MSG7: Tabela cadastrada com sucesso / Tabela atualizada com sucesso / Tabela excluída com sucesso.

A seguir têm-se as especificações dos casos de uso listados acima, bem como os seus cursos normais e alternativos.

Caso de Uso 1: CadastrarProjeto

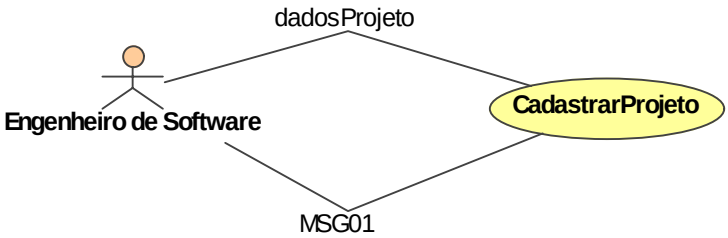


Figura 74 - Caso de Uso CadastrarProjeto

Curso Normal

1- Engenheiro informa que deseja fazer o cadastro do projeto

2- Sistema solicita os dados do projeto

3- Engenheiro informa nomeProjeto, descrição e observações

4- Sistema verifica que não existe o projeto cadastrado

5- Sistema cria uma instância de Projeto

6- Sistema solicita os dados da aplicação

7- Engenheiro informa nomeAplicacao, flgProcedural, flgOO e path

8- Sistema emite MSG01 informando que o projeto foi cadastrado com sucesso

Cursos Alternativos

4- Existe o projeto cadastrado

4.1 Sistema exibe o projeto

4.2 Sistema exibe opções para Alterar ou Excluir o projeto

4.3 Engenheiro escolhe a opção Alterar

4.4 Engenheiro atualiza os dados do projeto e confirma alteração

4.5 Sistema atualiza os dados do projeto

4.6 Sistema emite MSG01 informando que o projeto foi atualizado e encerra o Use Case

4.3 - Engenheiro escolhe a opção Excluir

4.3.1 Sistema pede confirmação da exclusão

4.3.2 Engenheiro confirma a exclusão

4.3.3 Sistema emite MSG01 informando que o projeto foi excluído e encerra o Use Case

Caso de Uso 2: Detalhar Projeto

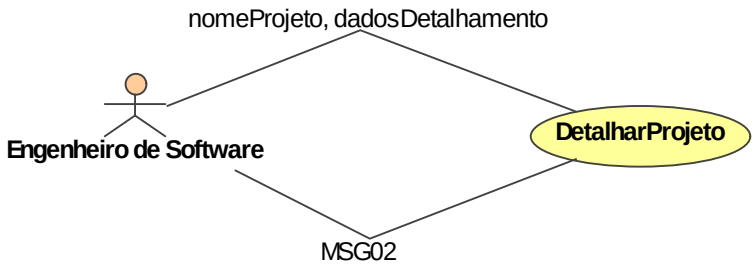


Figura 75 - Caso de Uso DetalharProjeto

Curso Normal

1- Engenheiro informa que deseja fazer o detalhamento do projeto

2- Sistema solicita os dados do detalhamento do projeto

3- Engenheiro informa nome do projeto, paths dos transformadores, observações

4- Sistema verifica se o projeto existe

5- Sistema cria uma instância de DetalhamentoProjeto

6- Sistema emite MSG02 informando que o detalhamento foi cadastrado com sucesso

Cursos Alternativos

4- Não existe o projeto cadastrado

4.1 Sistema chama o Use Case CadastrarProjeto e encerra o Use Case

Caso de Uso 3: ImportarModeloNegocio

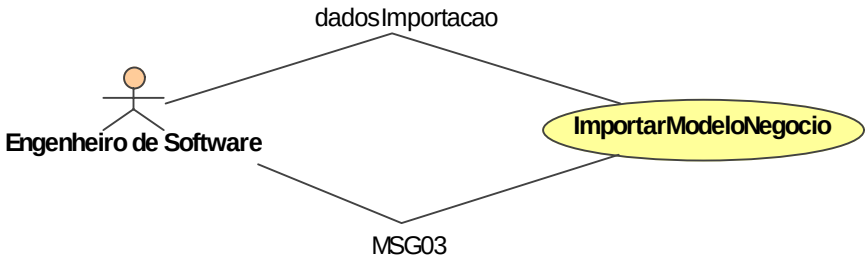


Figura 76 - Caso de Uso ImportarModeloNegocio

Curso Normal

1- Engenheiro informa que deseja fazer a importação do modelo de negócio

2- Sistema solicita os dados da importação

3- Engenheiro informa nomeProjeto, nomeModulos, data, flgTabelasBasicas, versao, pathXML e observações

4- Sistema verifica se o projeto existe

5- Sistema verifica se o módulo existe

6- Sistema cria uma instância de GerenciaImportacaoXML

7- Sistema emite MSG03 informando que a importação foi realizada com sucesso

Cursos Alternativos

4- Não existe o projeto cadastrado

4.1 Sistema solicita o Use Case CadastrarProjeto e encerra o Use Case

5- Não existe o módulo cadastrado

5.1 Sistema solicita o Use Case CadastrarModulo e encerra o Use Case

Caso de Uso 4: DefinirEtapasDeMigracao

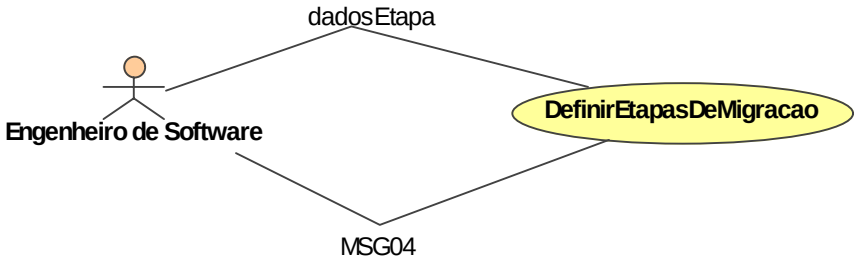


Figura 77 - Caso de Uso DefinirEtapasDeMigracao

Curso Normal

1- Engenheiro informa que deseja definir as etapas do projeto de migração

2- Sistema solicita os dados da etapa

3- Engenheiro informa nomeProjeto, nomeEtapa, descrição, dataPlanejamento, dataInicioExecução, dataFinalExecução, nomeModulo, nomeProgramas

4- Sistema verifica se o projeto existe

5- Sistema verifica se a etapa existe

6- Sistema verifica se o módulo existe

7- Sistema verifica se o programa existe

8- Sistema cria uma instância de Etapa

9- Sistema emite MSG04 informando que etapa foi cadastrada com sucesso

Cursos Alternativos

4- Não existe o projeto cadastrado

4.1 Sistema solicita o Use Case CadastrarProjeto e encerra o Use Case

5- Existe a etapa cadastrada

5.1 Sistema exibe a etapa

5.2 Sistema exibe opções para Alterar ou Excluir a etapa

5.3 Engenheiro escolhe a opção Alterar

5.4 Engenheiro atualiza os dados da etapa e confirma alteração

5.5 Sistema atualiza os dados da etapa

5.6 Sistema emite MSG06 informando que a etapa foi atualizada e encerra o Use Case

5.3 - Engenheiro escolhe a opção Excluir

5.3.1 Sistema pede confirmação da exclusão

6.3.2 Engenheiro confirma a exclusão

5.3.3 Sistema emite MSG06 informando que a etapa foi excluída e encerra o Use Case

6- Não existe o módulo cadastrado

8.1 Sistema solicita o Use Case CadastrarModulo e encerra o Use Case

7- Não existe o programa cadastrado

7.3 Sistema solicita o Use Case CadastrarPrograma e encerra o Use Case

Caso de Uso 5: ExportarDescricoesEtapa

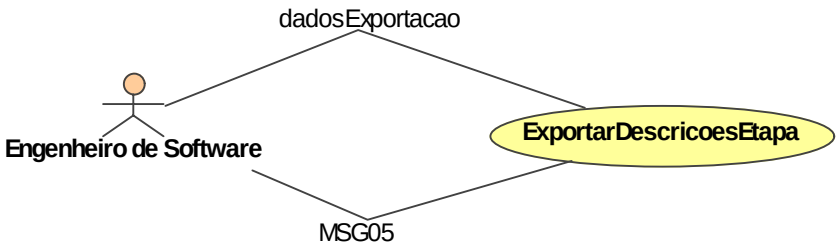


Figura 78 – Caso de Uso ExportarDescricoesEtapa

Curso Normal

1- Engenheiro informa que deseja fazer a exportação das definições da etapa

2- Sistema solicita os dados da exportação

3- Engenheiro informa nomeProjeto, nomeEtapa,, dataExportacao

4- Sistema verifica se o projeto existe

5- Sistema verifica se a etapa existe

6- Sistema cria uma instância de GerenciaExportacaoXML

7- Sistema emite MSG05 informando que etapa foi exportada com sucesso

Cursos Alternativos

4- Não existe o projeto cadastrado

4.1 Sistema solicita o Use Case CadastrarProjeto e encerra o Use Case

5- Não existe a etapa cadastrada

5.1 Sistema solicita o Use Case DefinirEtapasDeMigracao e encerra o Use Case

Caso de Uso 6: ConsultarDadosInterface

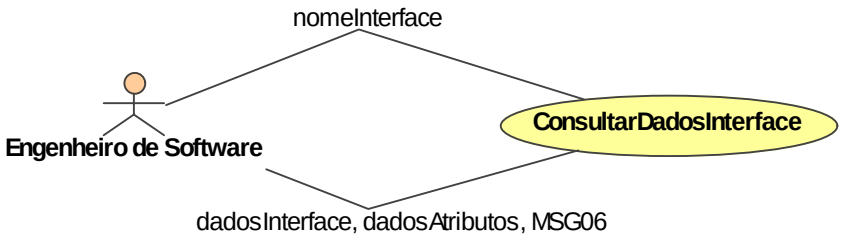


Figura 79 - Caso de Uso ConsultarDadosInterface

Curso Normal

1- Engenheiro informa que deseja consultar interfaces

2- Sistema solicita o nome da interface

3- Engenheiro informa o nome da interface

4- Sistema verifica que a interface existe

6- Sistema mostra dados da interface e seus atributos

Curso Alternativo

4-Não existe a interface cadastrada

4.3 Sistema emite MSG09 informando que interface não existe

4.4 Sistema exibe opção para Incluir Interface

4.5 Engenheiro inclui Interface

4.6 Sistema emite MSG06 informando que a interface foi cadastrada com sucesso

Caso de Uso 7: ConsultarDadosTabela

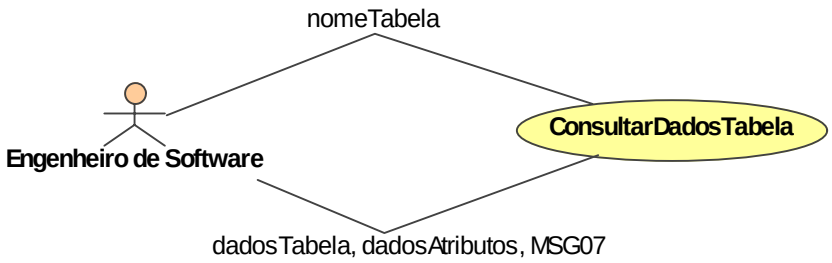


Figura 80 - Caso de Uso ConsultarDadosTabela

Curso Normal

1- Engenheiro informa que deseja consultar tabelas

2- Sistema solicita o nome da tabela

3- Engenheiro informa o nome da tabela

4- Sistema verifica que a tabela existe

6- Sistema mostra dados da tabela e seus atributos

Curso Alternativo

4-Não existe a tabela cadastrada

4.3 Sistema emite MSG07 informando que a tabela não existe

4.4 Sistema exibe opção para Incluir Tabela

4.5 Engenheiro inclui Tabela

4.6 Sistema emite MSG07 informando que a tabela foi cadastrada com sucesso

2 Modelos de Seqüência

A seguir são apresentados os modelos de seqüência para os casos de uso.

Modelo de Seqüência: CadastrarProjeto

(Curso Normal)

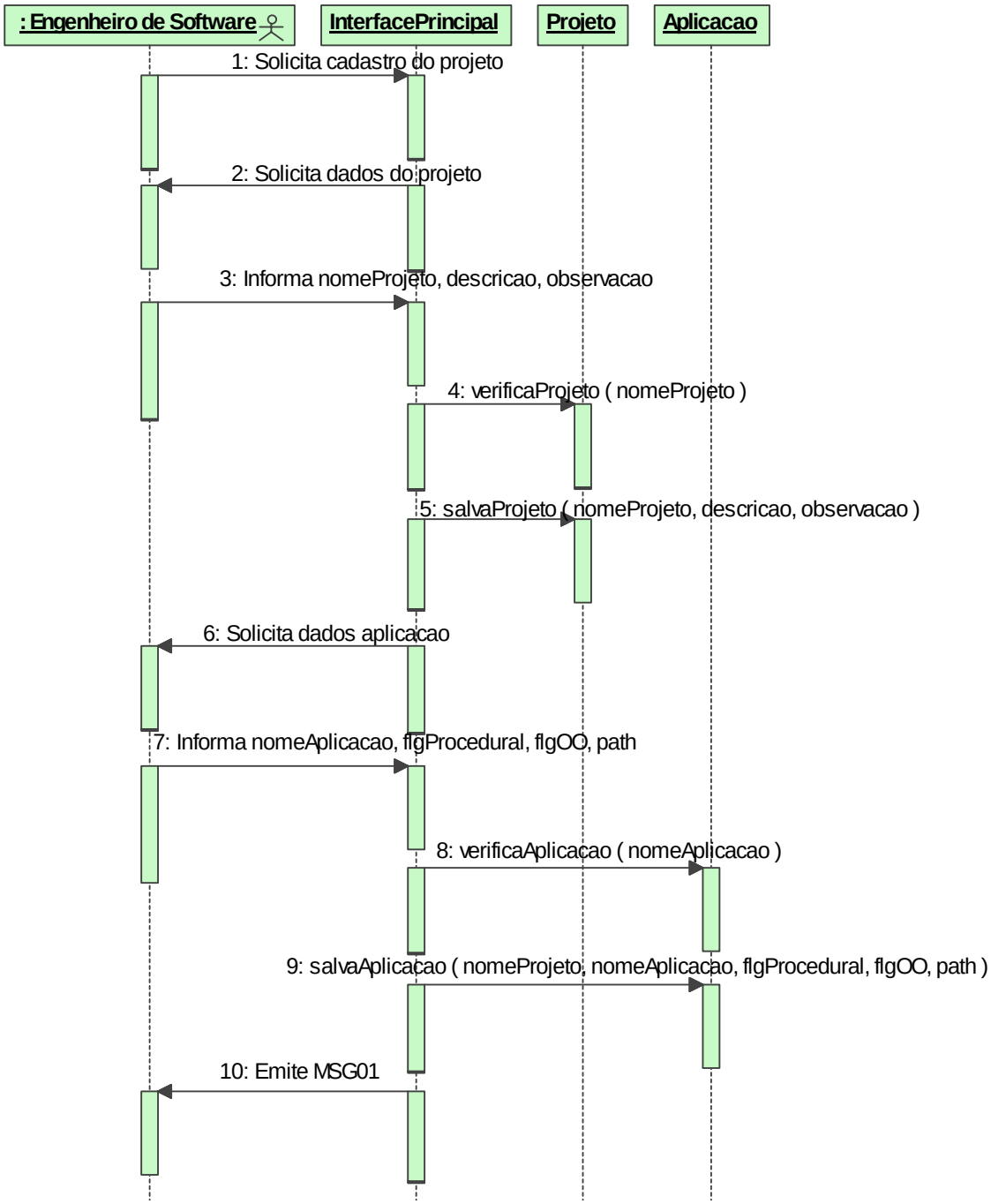


Figura 81 - Modelo de Seqüência CadastrarProjeto (Curso Alternativo 4)

(Curso Alternativo 4)

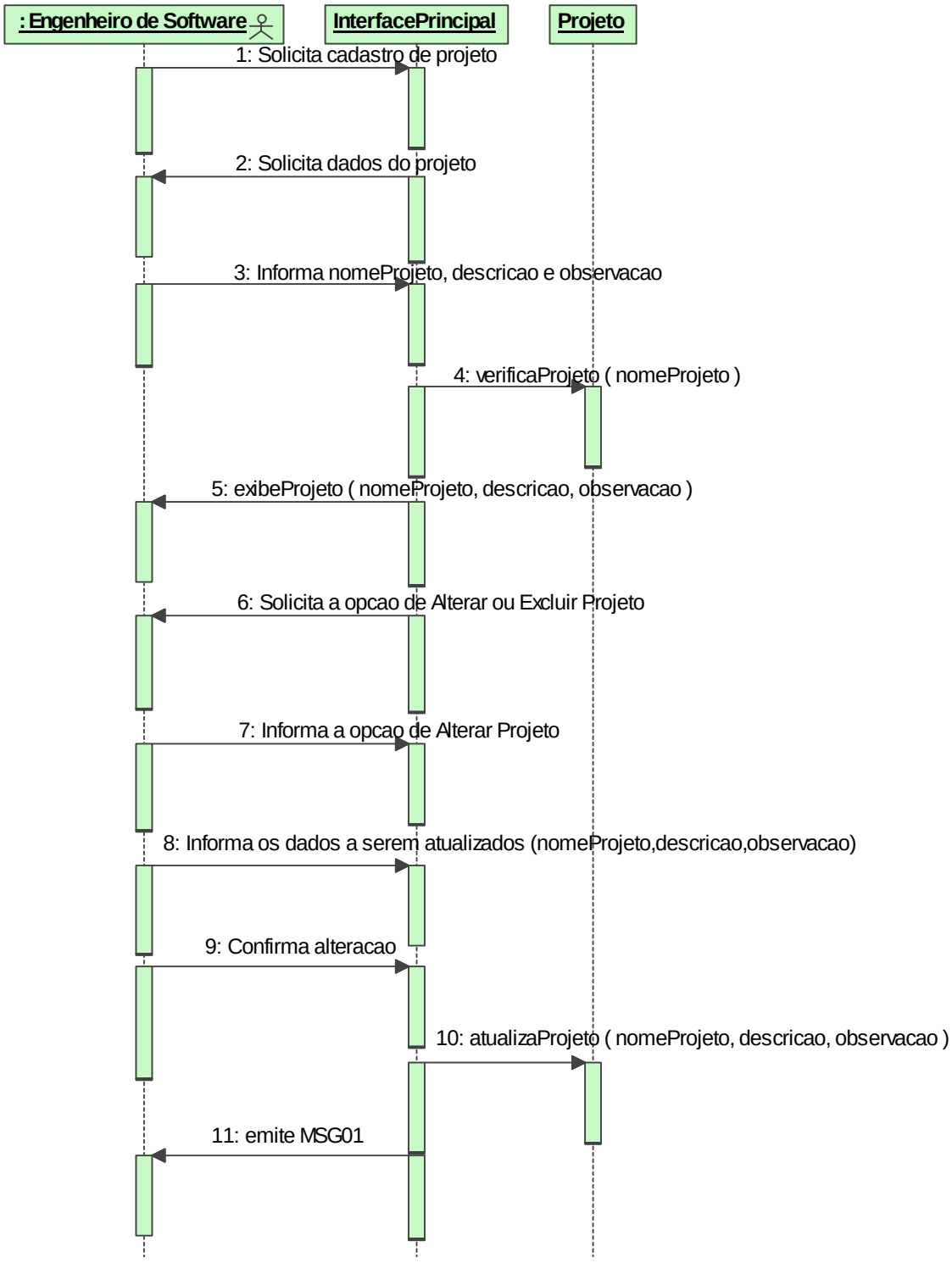


Figura 82 – Modelo de Seqüência CadastrarProjeto (Curso Alternativo 4)

(Curso Alternativo 4.3)

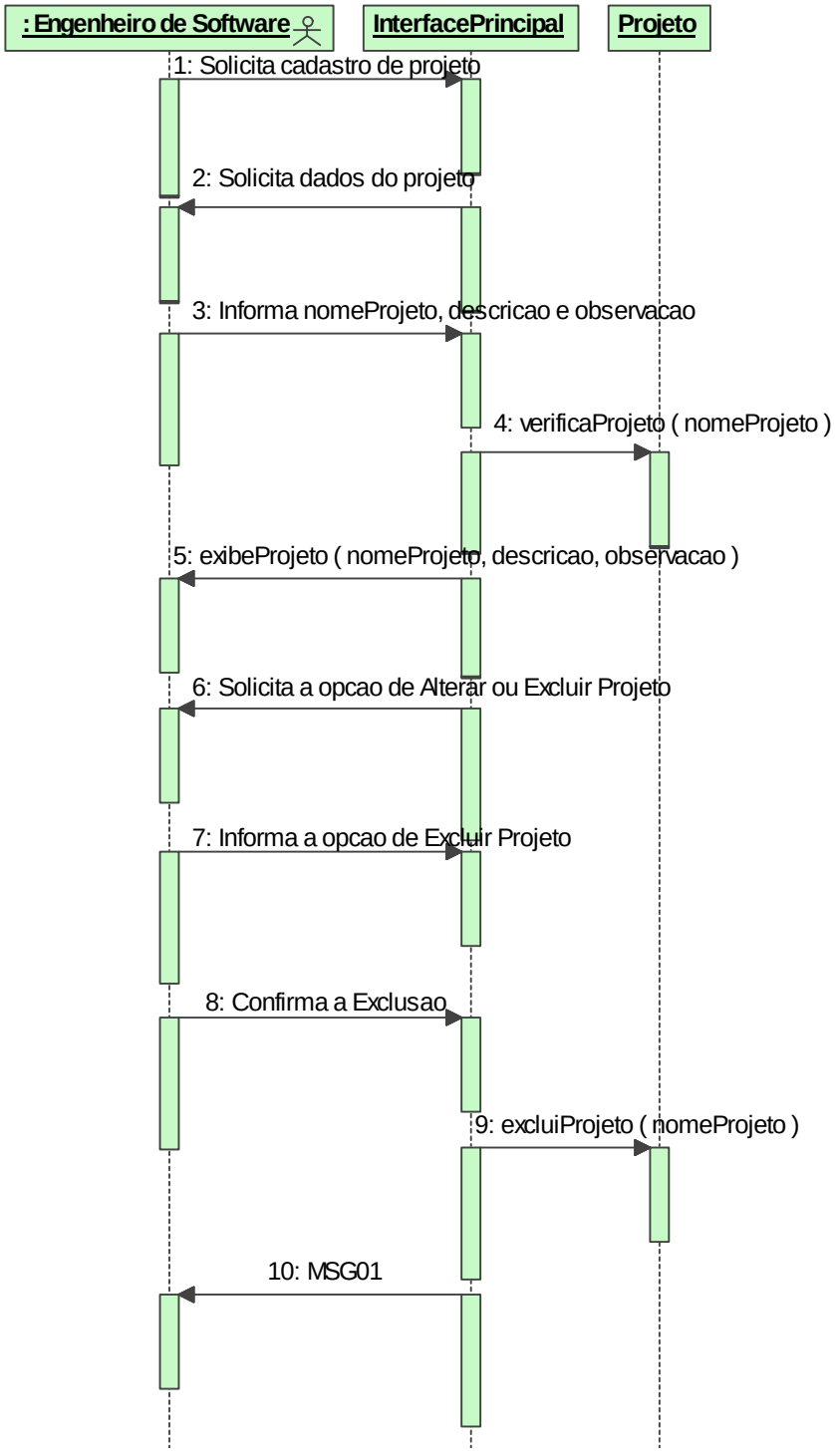


Figura 83 – Modelo de Seqüência CadastrarProjeto (Curso Alternativo 4.3)

Modelo de Seqüência: DetalharProjeto

(Curso Normal)

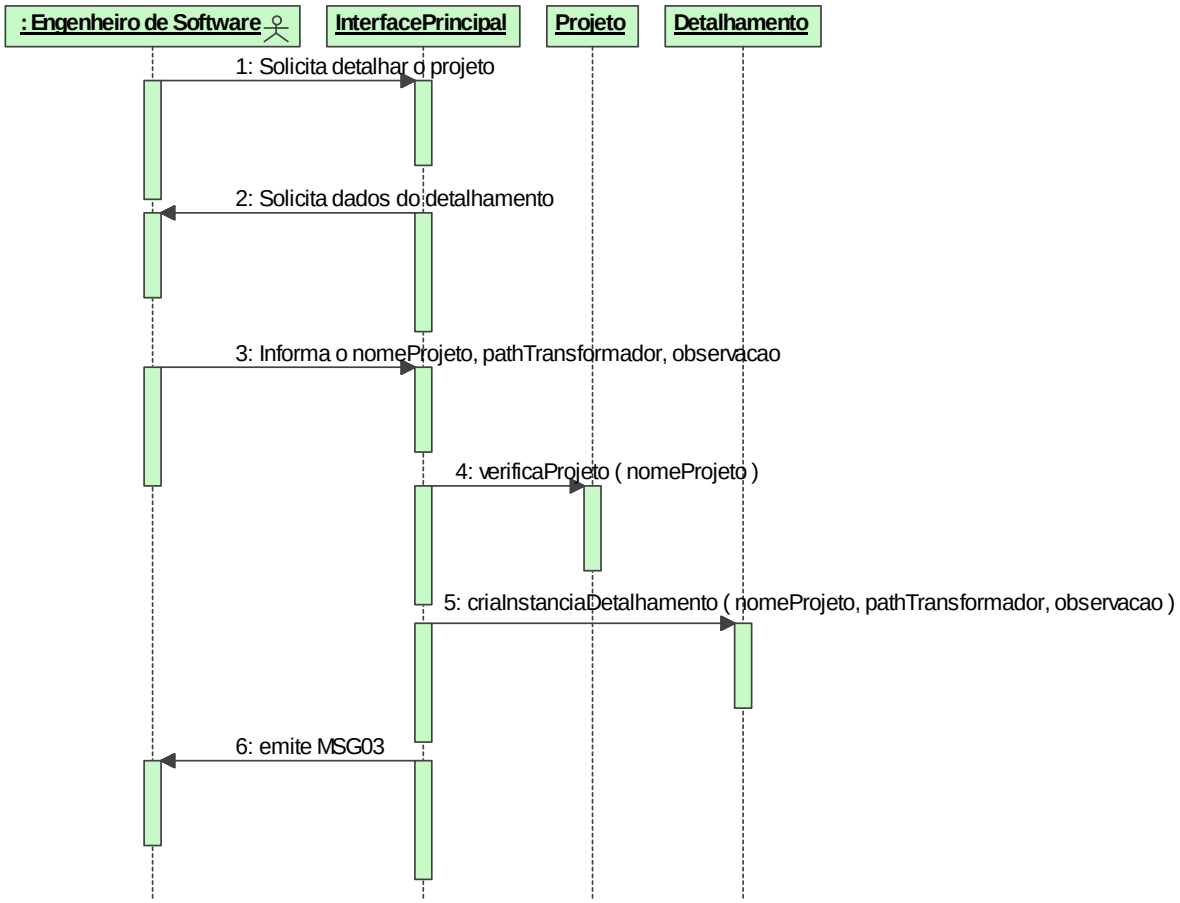


Figura 84 – Modelo de Seqüência DetalharProjeto (Curso Normal)

(Curso Alternativo 4)

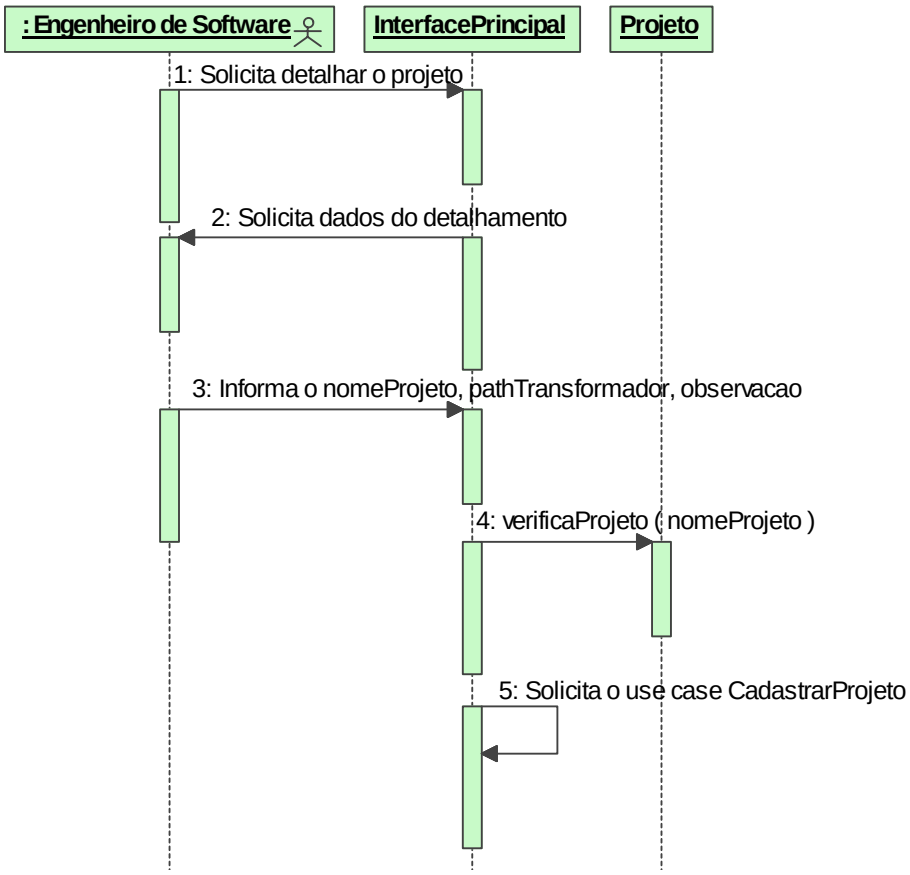


Figura 85 - Modelo de Seqüência DetalharProjeto (Curso Alternativo 4)

Modelo de Seqüência: ImportarModeloNegocio

(Curso Normal)

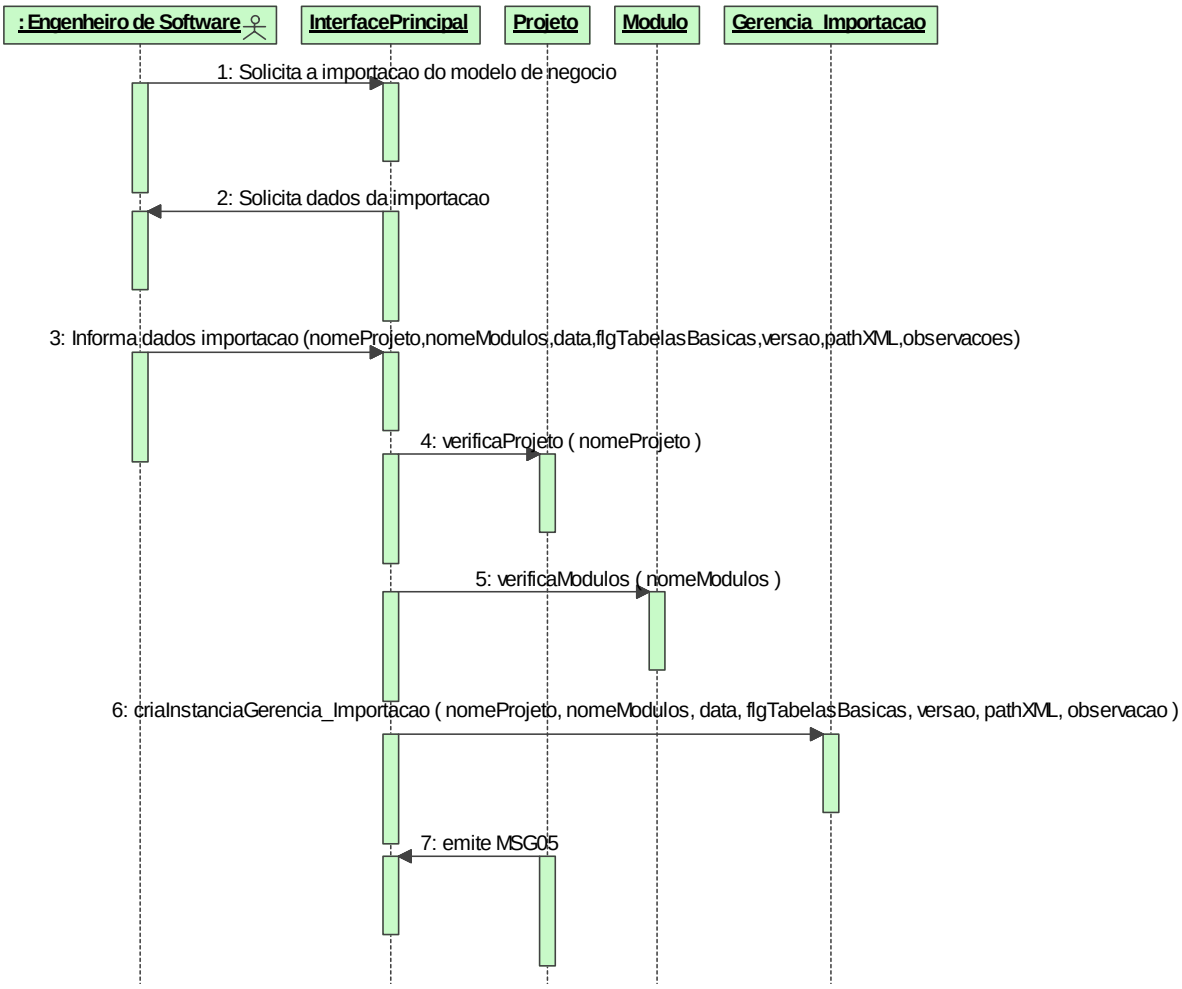


Figura 86 - Modelo de Seqüência ImportarModeloNegocio (Curso Normal)

(Curso Alternativo 4)

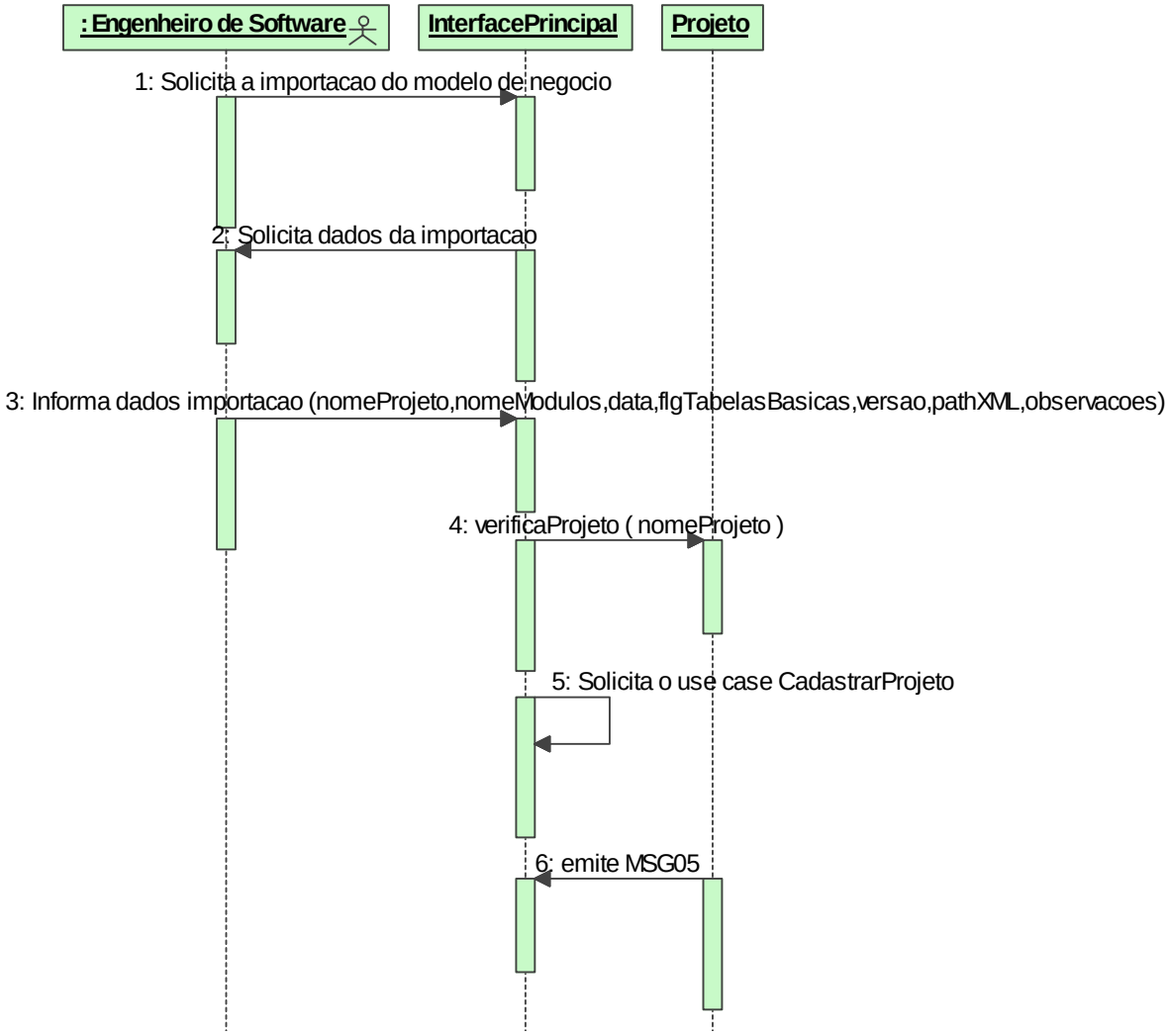


Figura 87 - Modelo de Seqüência ImportarModeloNegocio (Curso Alternativo 4)

(CursoAlternativo 5)

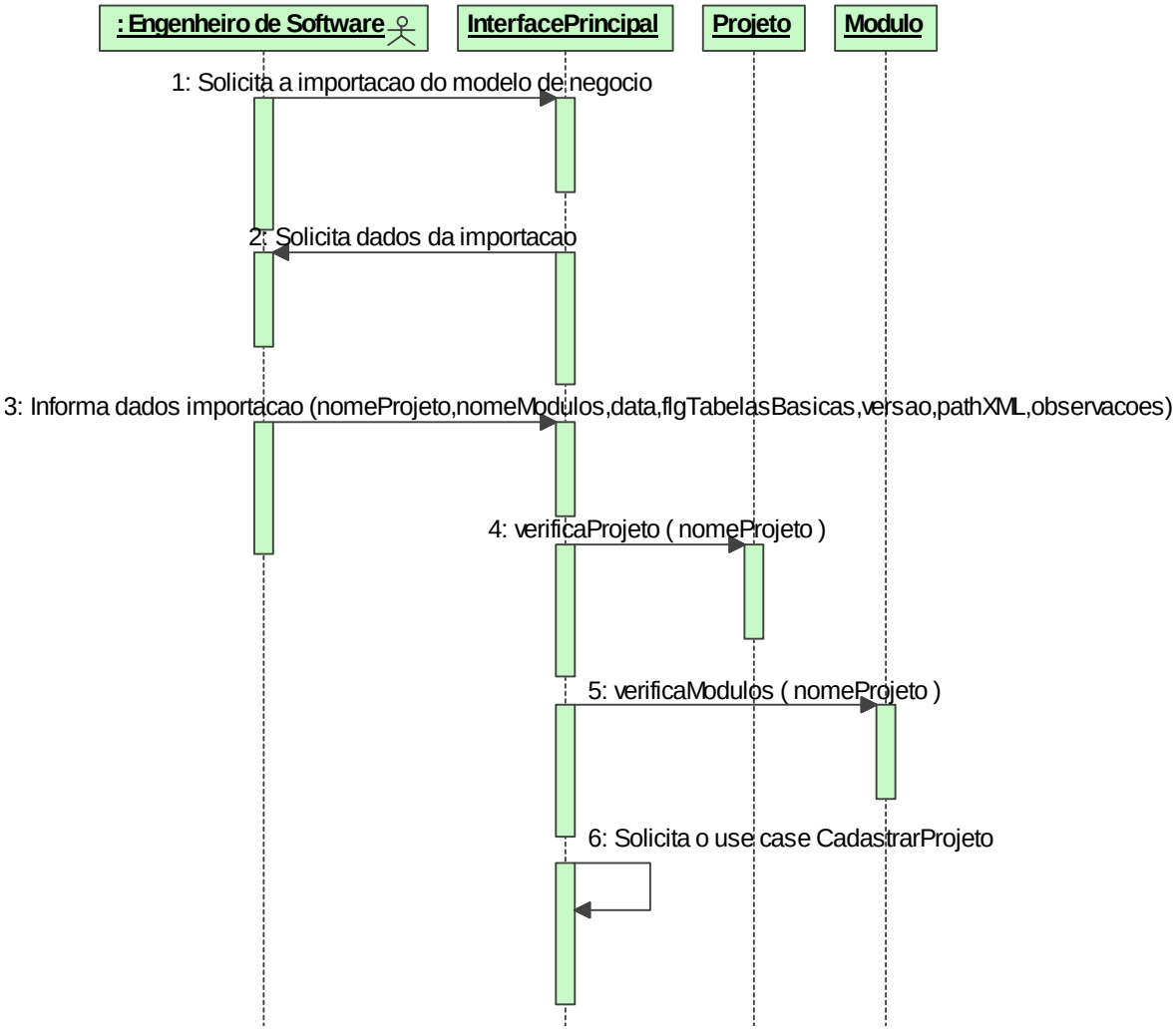


Figura 88 - Modelo de Seqüência ImportarModeloNegocio (Curso Alternativo 5)

Modelo de Seqüência: DefinirEtapasDeMigracao

(Curso Normal)

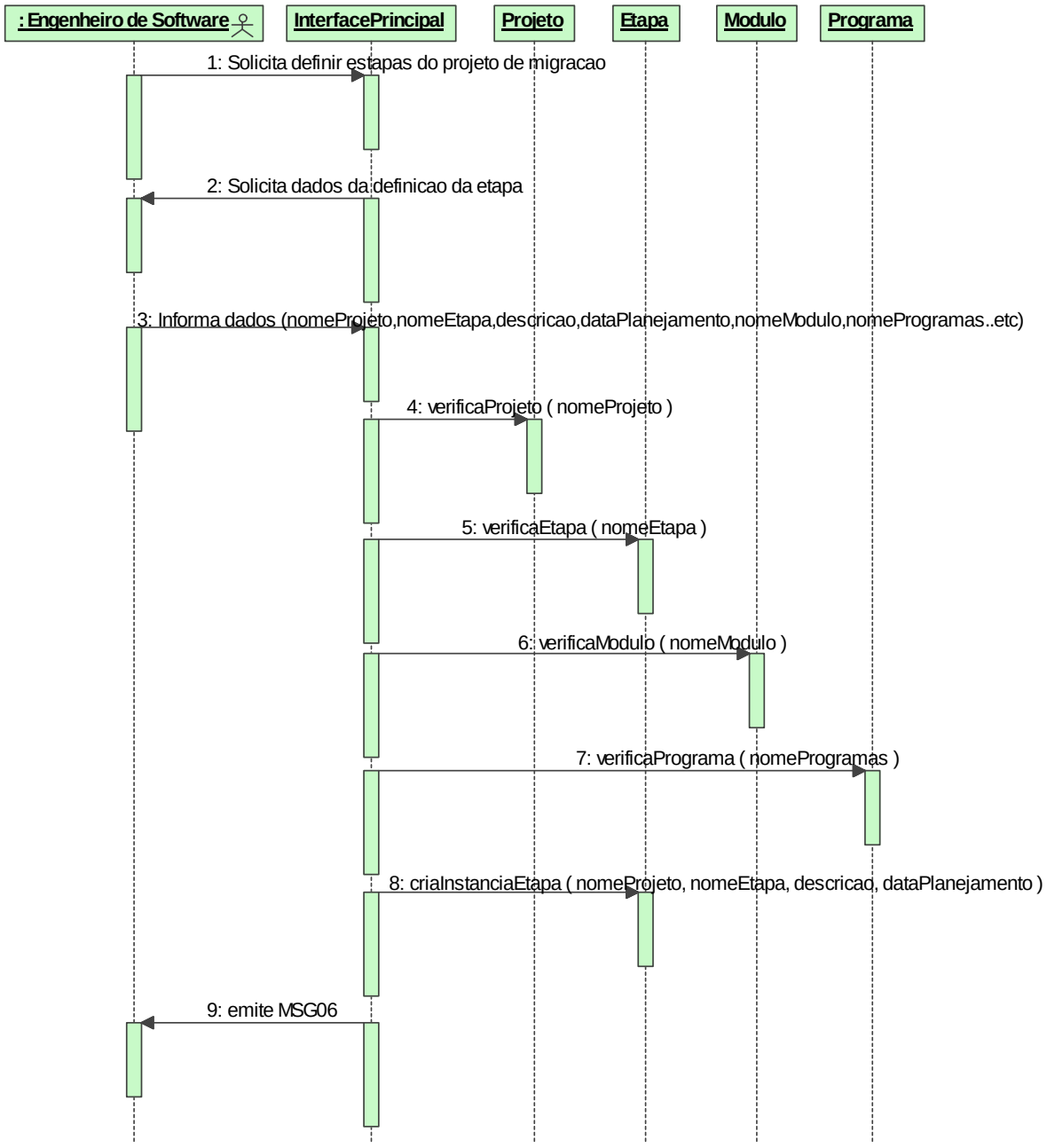


Figura 89 – Modelo de Seqüência DefinirEtapasDeMigracao (Curso Normal)

(Curso Alternativo 4)

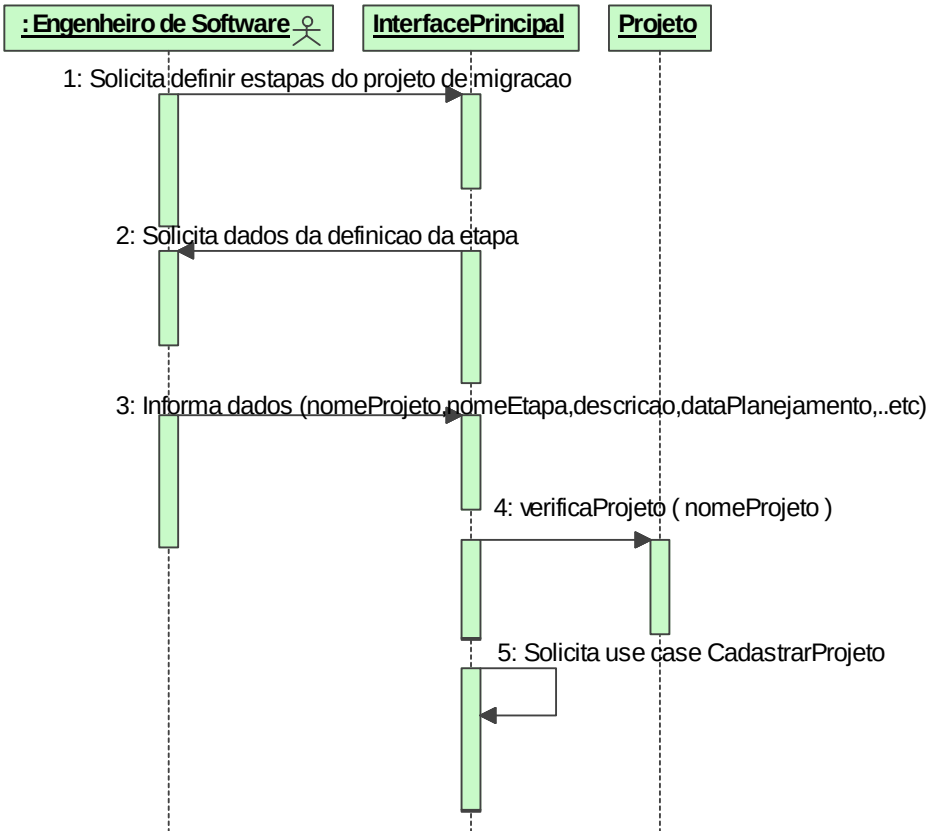


Figura 90 - Modelo de Seqüência DefinirEtapasDeMigracao (Curso Alternativo 4)

(Curso Alternativo 5)

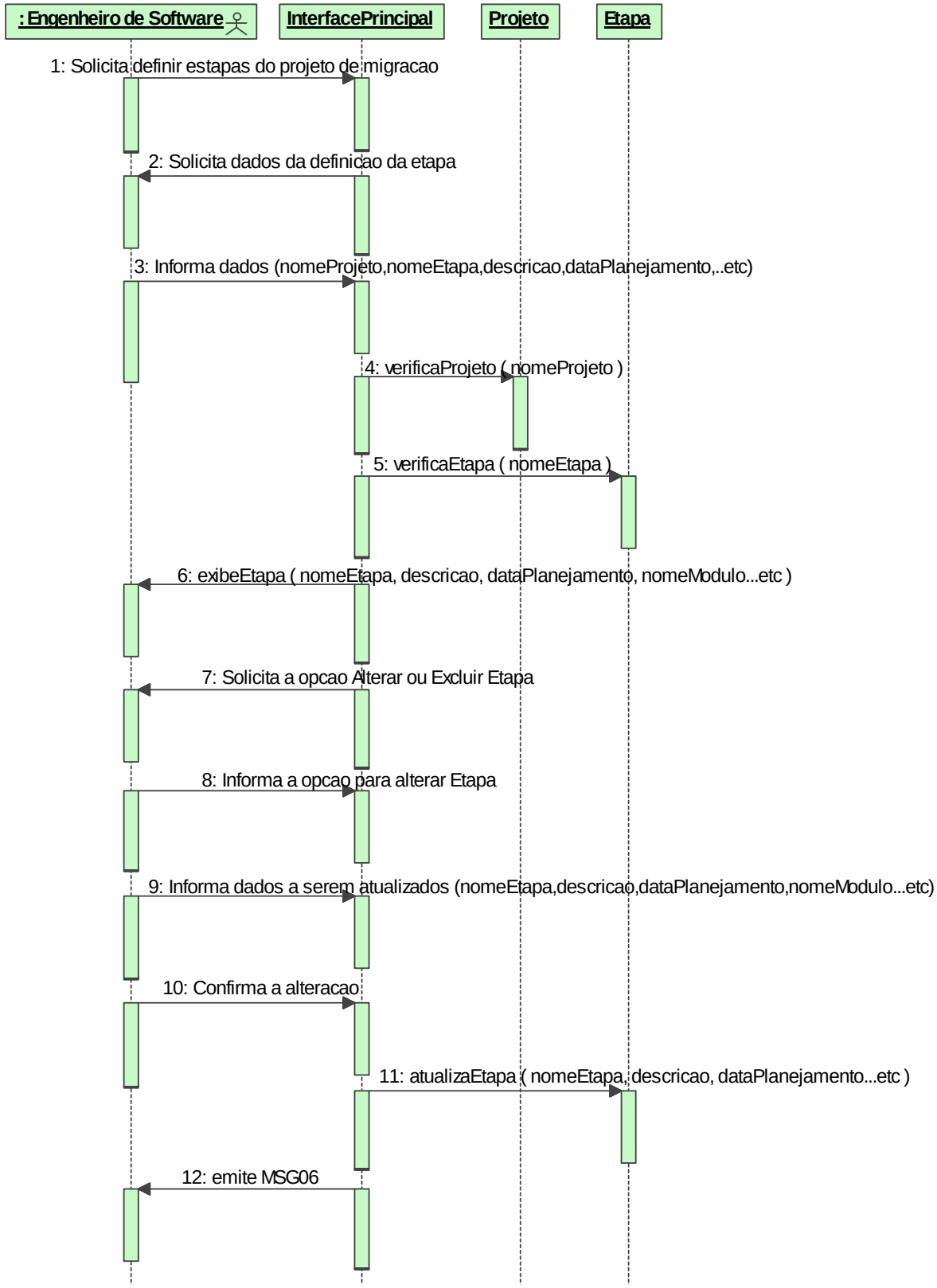


Figura 91 - Modelo de Seqüência DefinirEtapasDeMigracao (Curso Alternativo 5)

(Curso Alternativo 5.3)

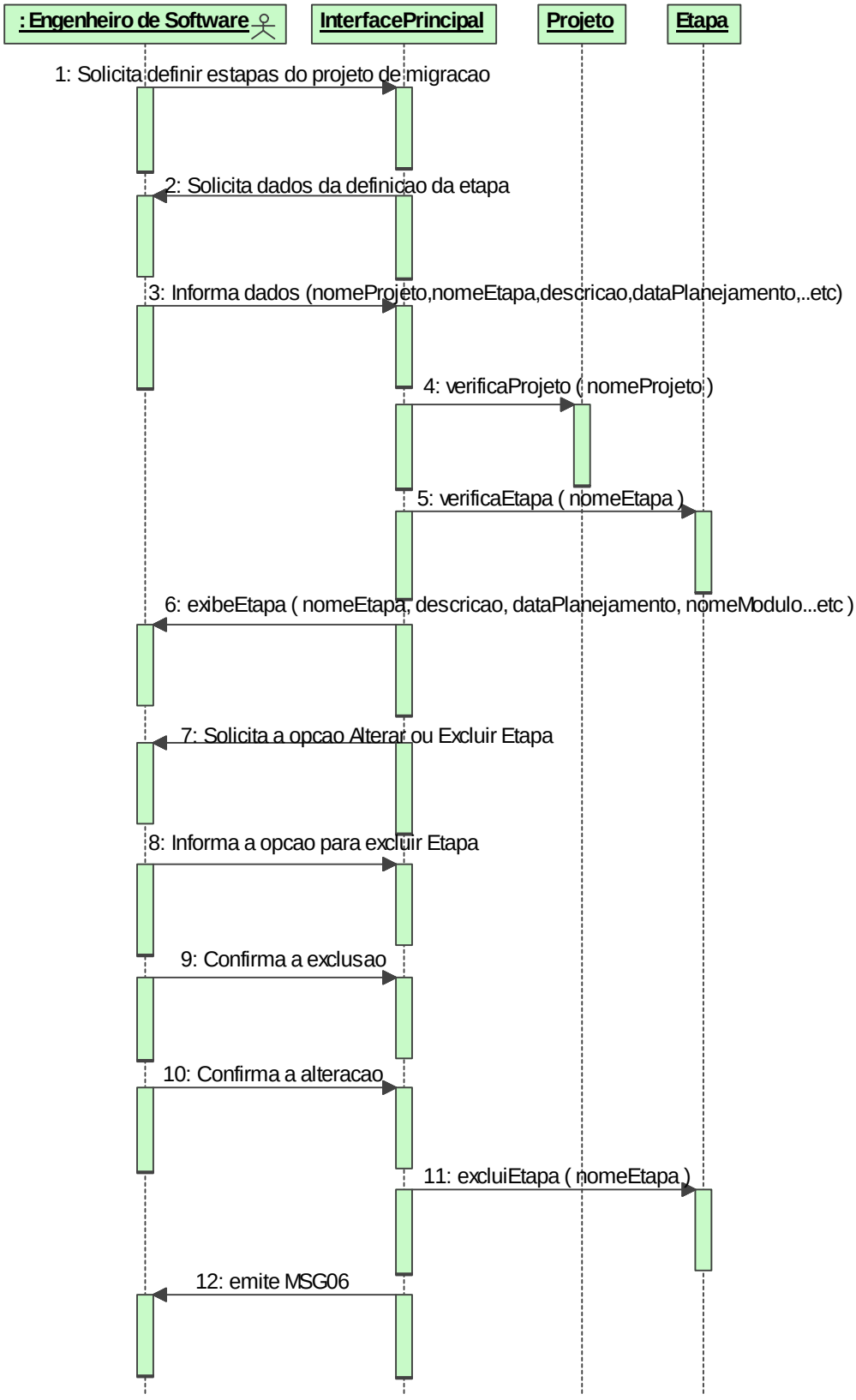


Figura 92 - Modelo de Seqüência DefinirEtapasDeMigracao (Curso Alternativo 5.3)

(Curso Alternativo 6)

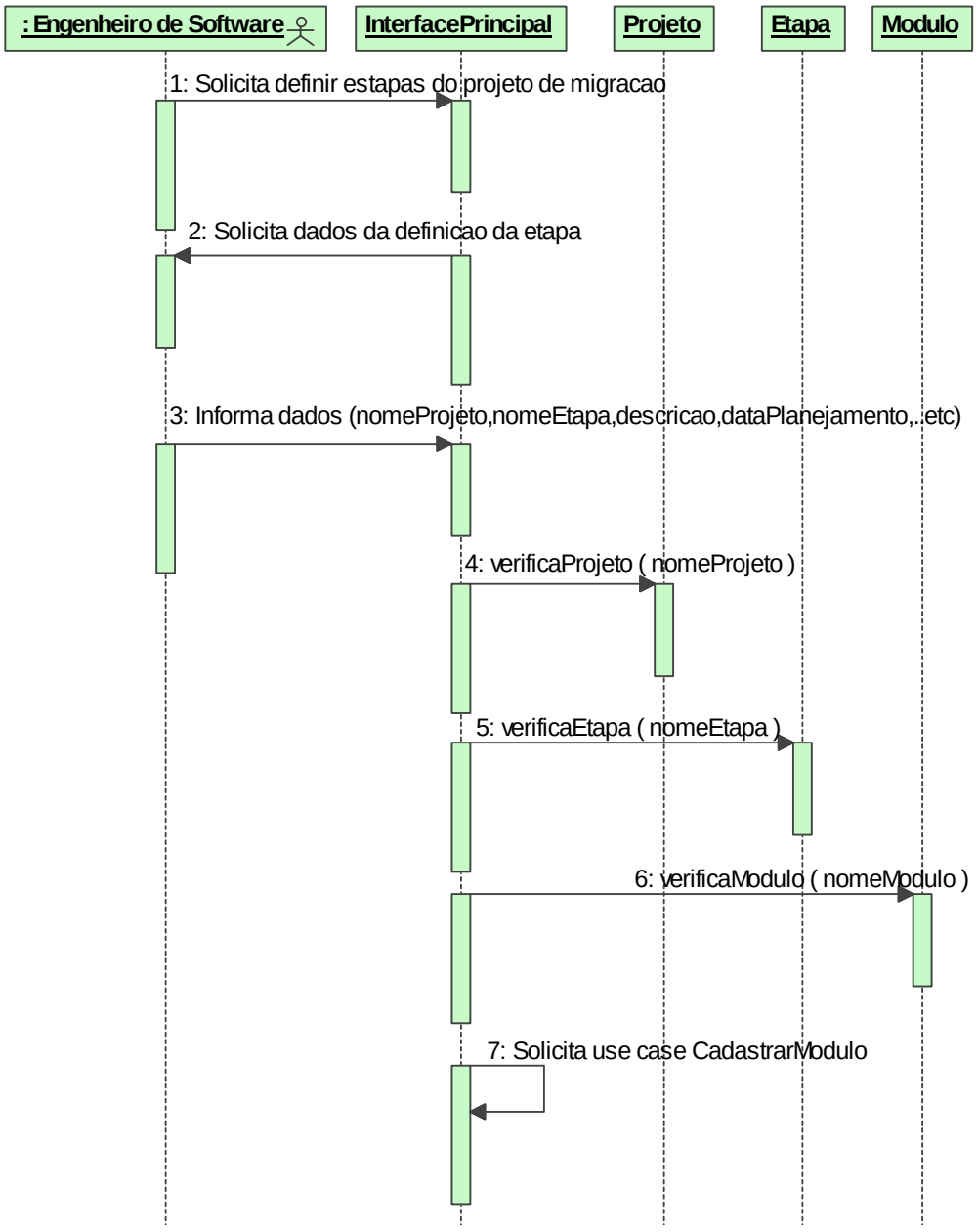


Figura 93 - Modelo de Seqüência DefinirEtapasDeMigracao (Curso Alternativo 6)

(Curso Alternativo 7)

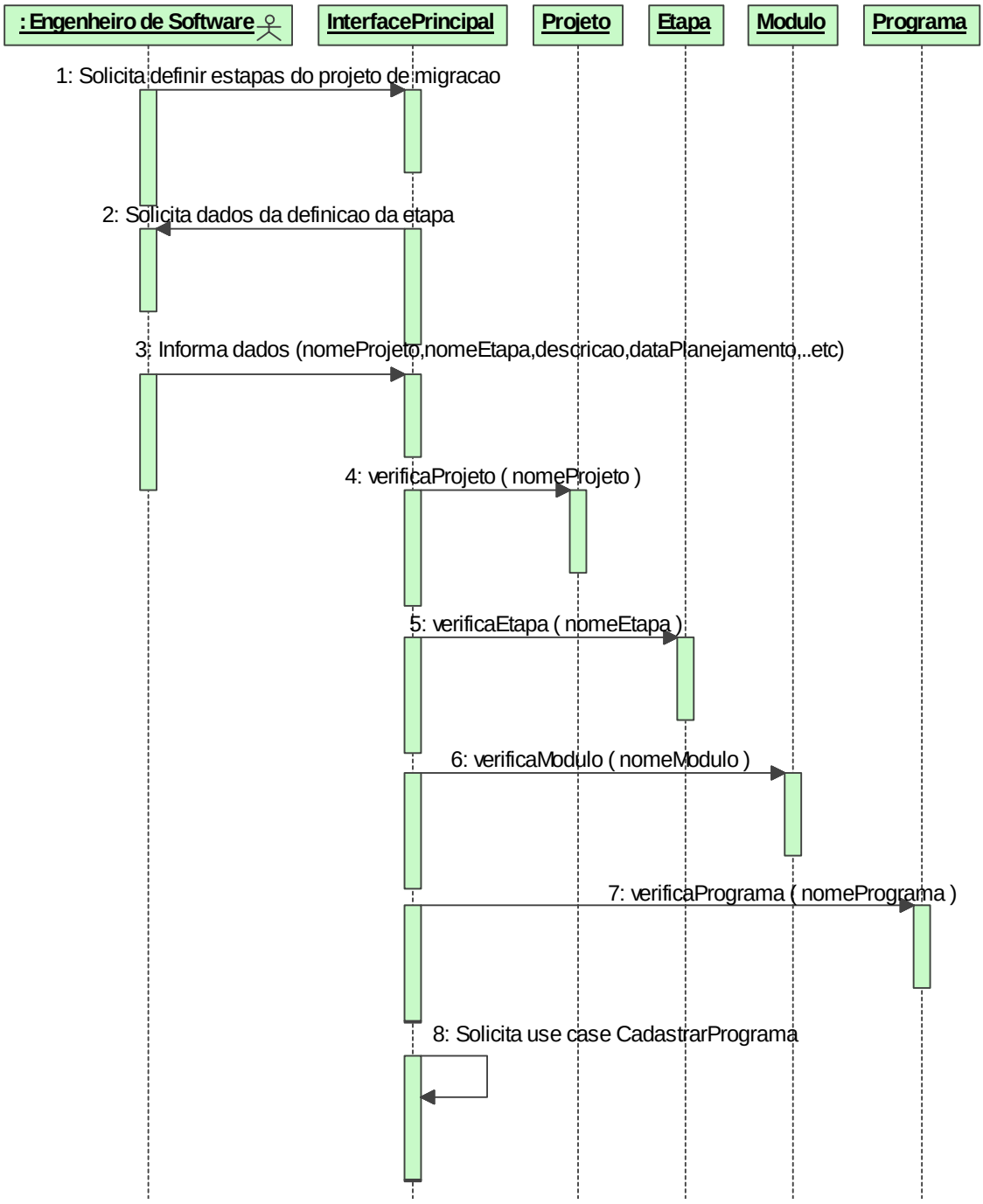


Figura 94 - Modelo de Seqüência DefinirEtapasDeMigracao (Curso Alternativo 7)

Modelo de Seqüência: ExportarDescricoesEtapa

(Curso Normal)

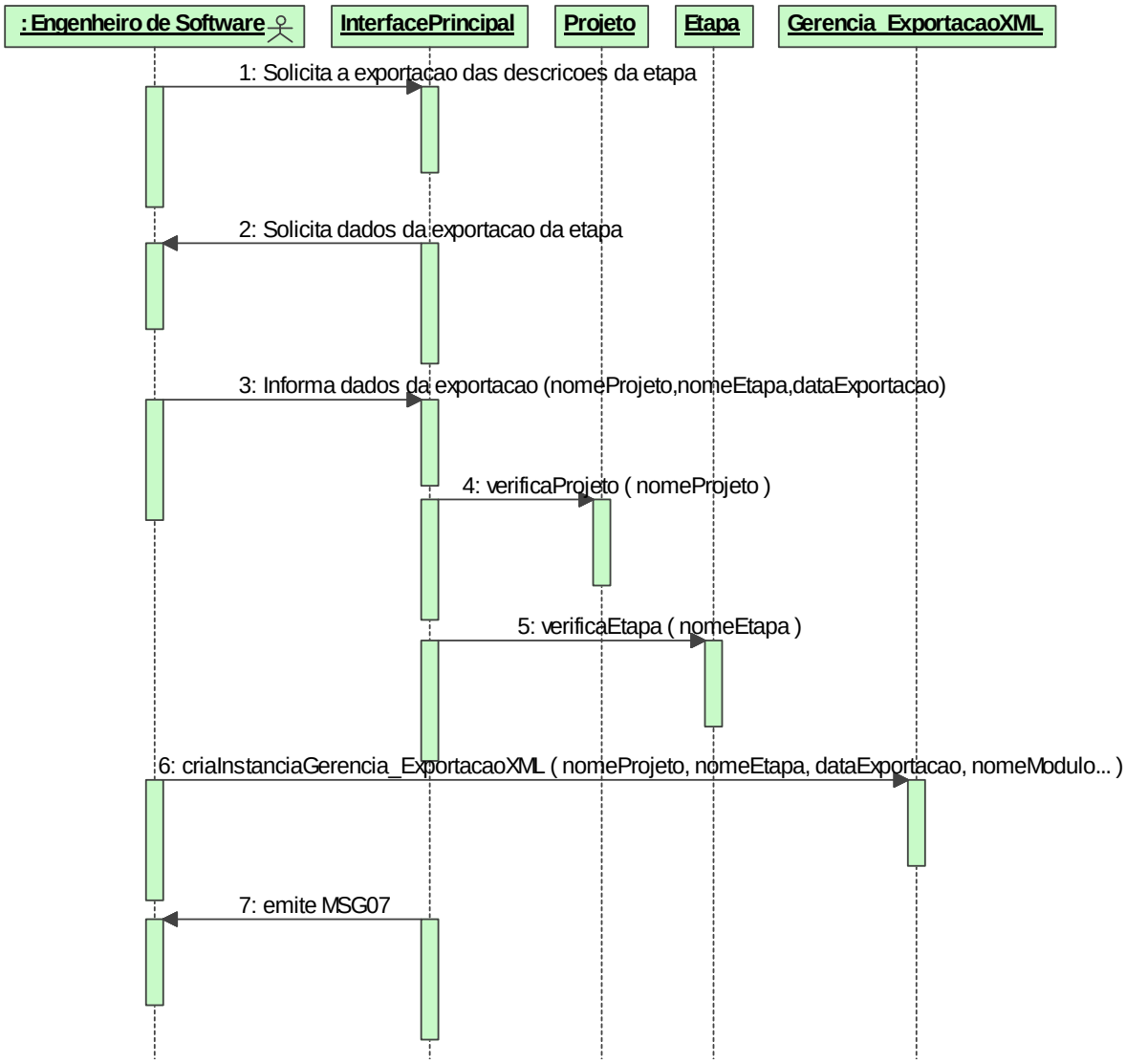


Figura 95 - Modelo de Seqüência ExportarDescricoesEtapa (Curso Normal)

(Curso Alternativo 4)

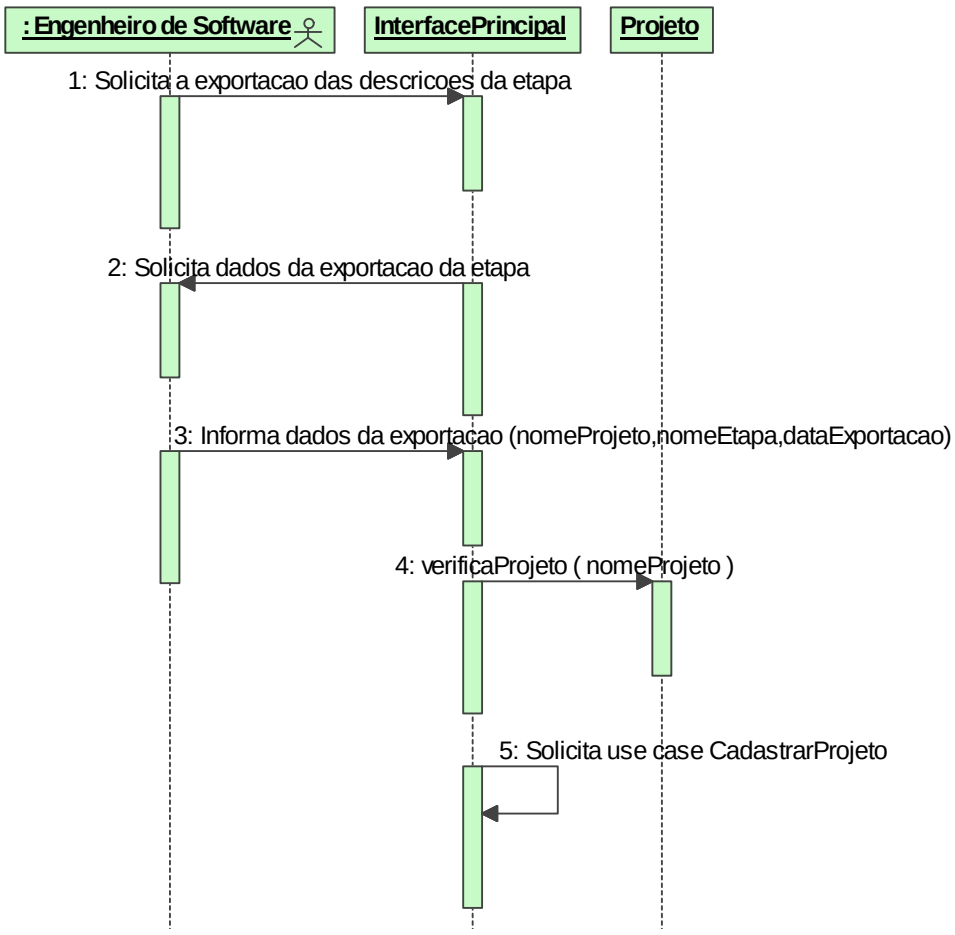


Figura 96 - Modelo de Seqüência ExportarDescricoesEtapa (Curso Alternativo 4)

(Curso Alternativo 5)

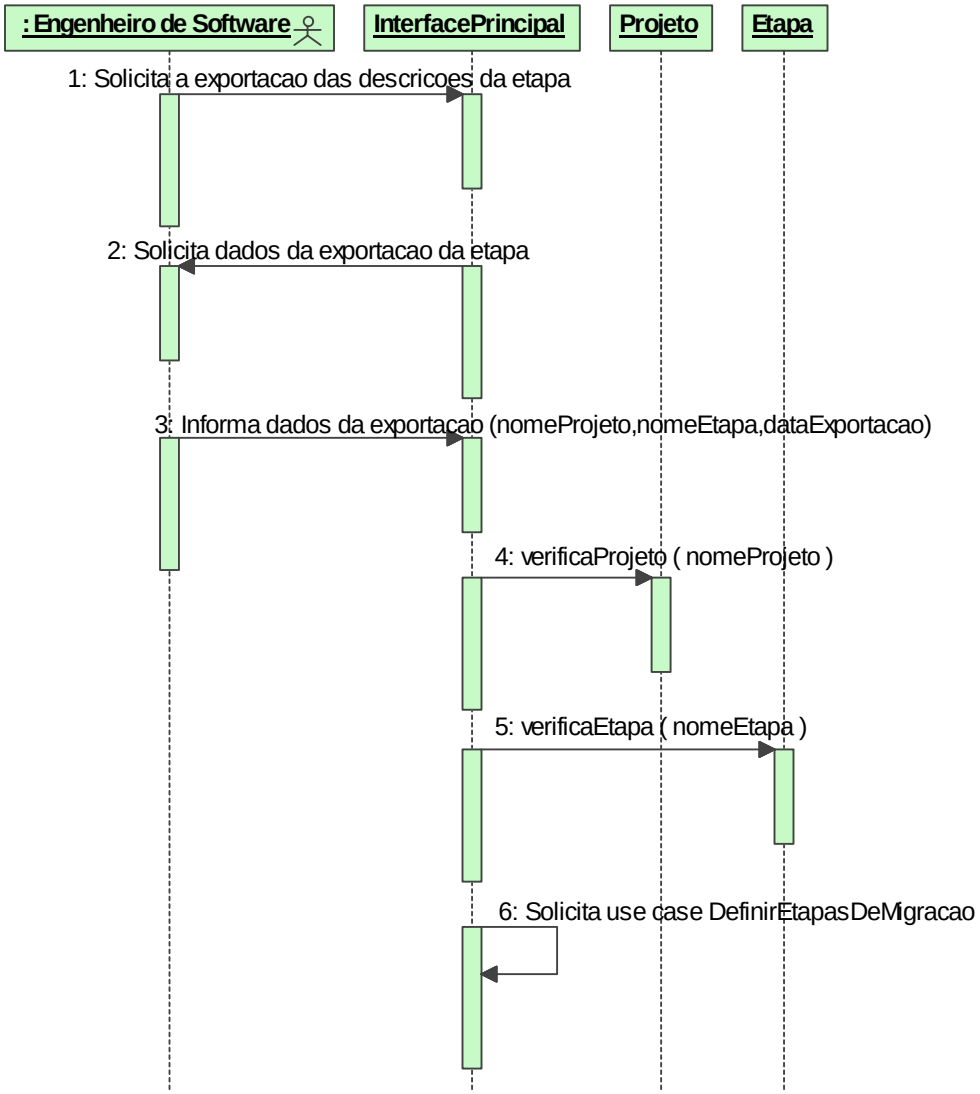


Figura 97 - Modelo de Seqüência ExportarDescricoesEtapa (Curso Alternativo 5)

Modelo de Seqüência: ConsultarDadosInterface

(Curso Normal)

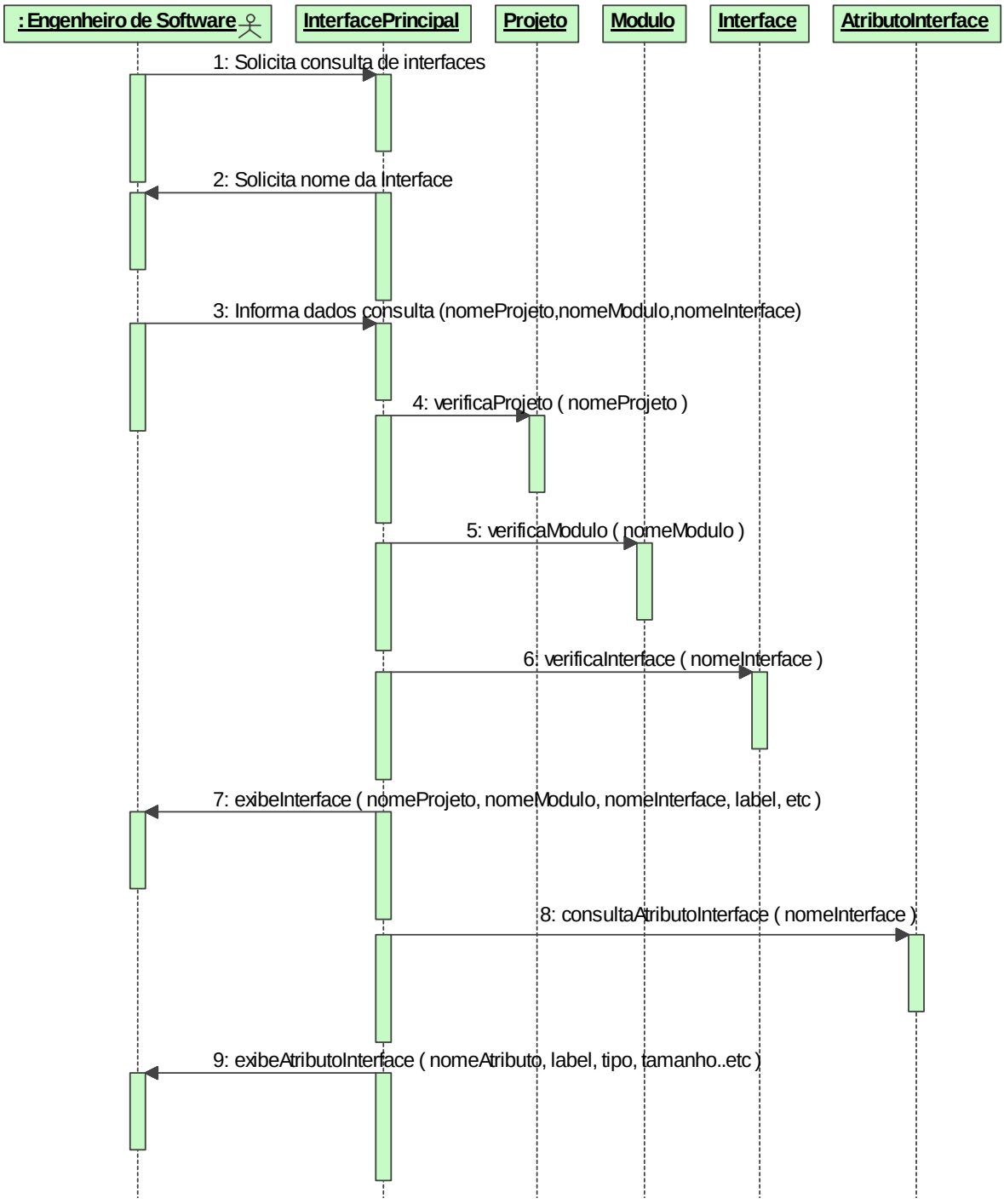


Figura 98 - Modelo de Seqüência ConsultarDadosInterface (Curso Normal)

Modelo de Seqüência: ConsultarDadosTabela

(Curso Normal)

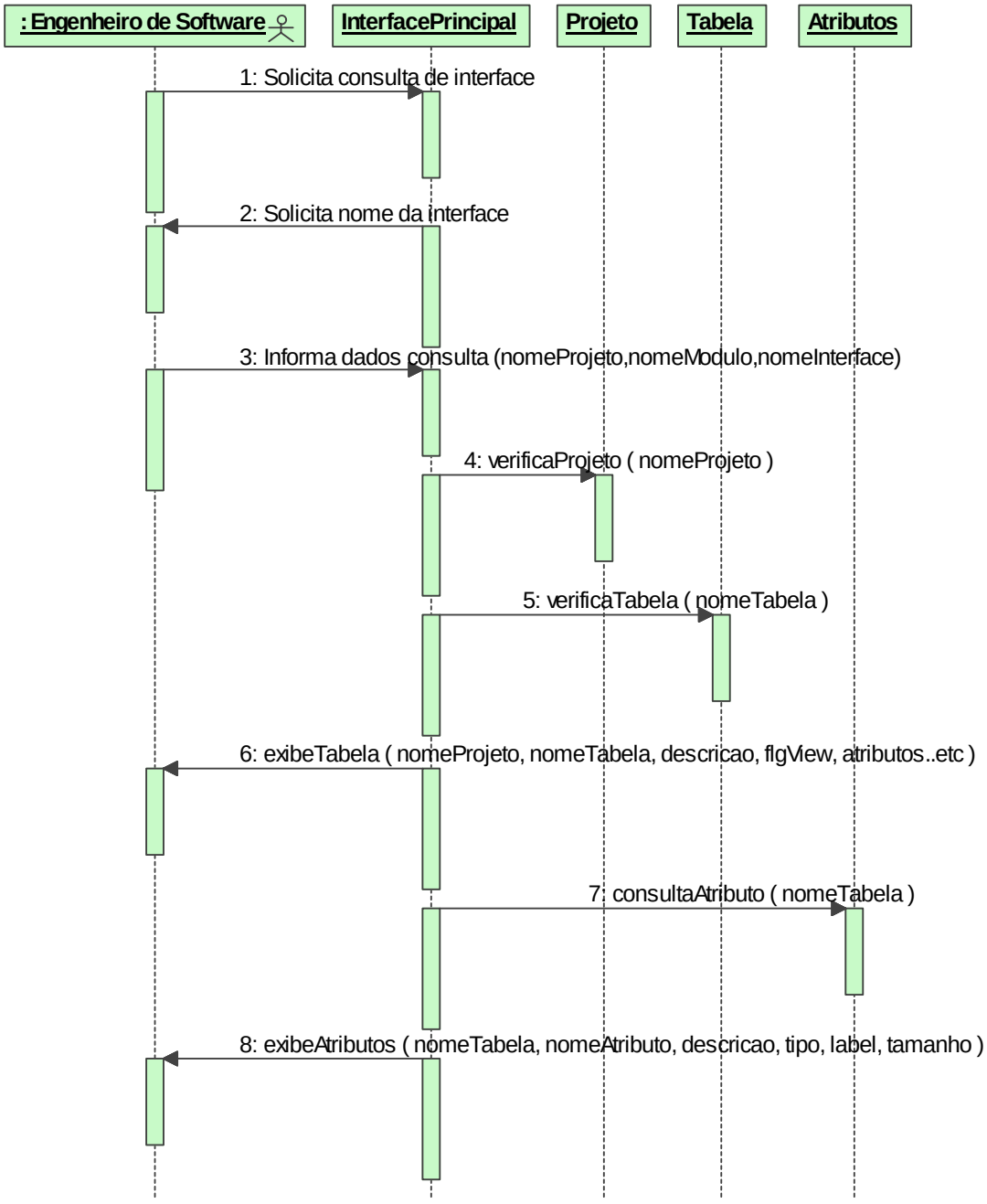


Figura 99 - Modelo de Seqüência ConsultarDadosTabela (Curso Normal)

3 Documentação gerada pelo Apyon Studio

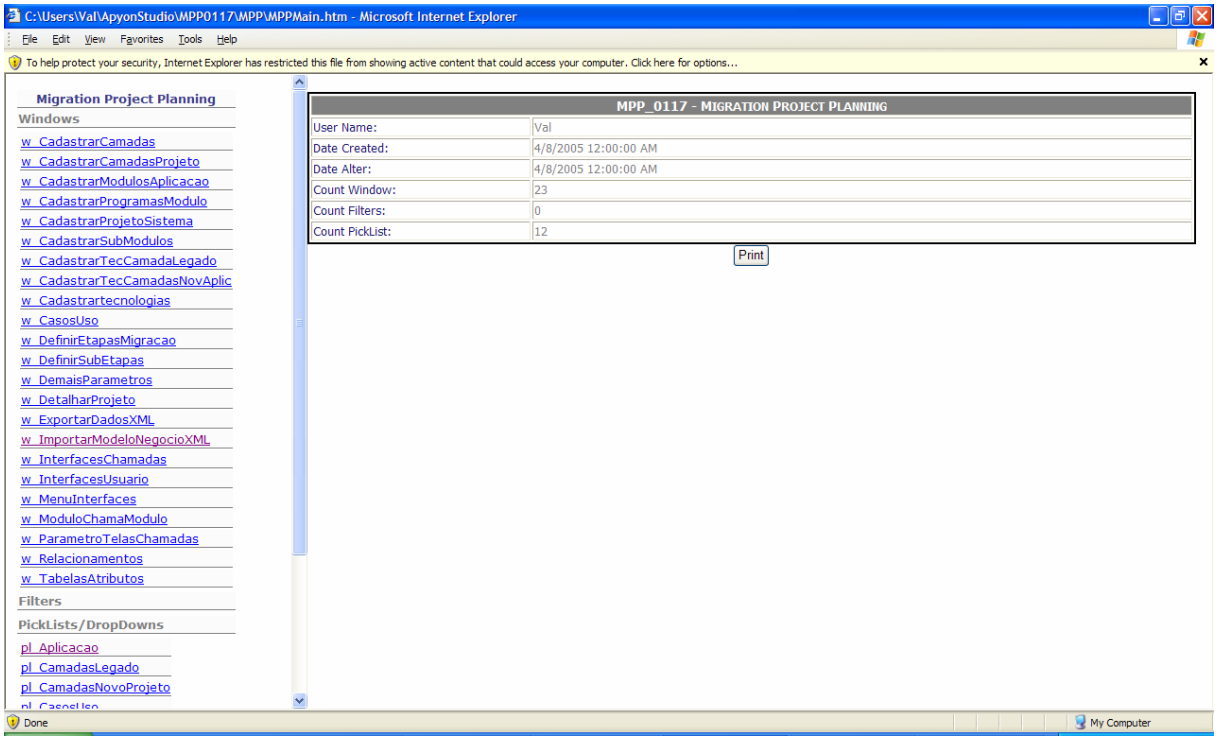


Figura 100 – Relatório de documentação do Menu do MPP

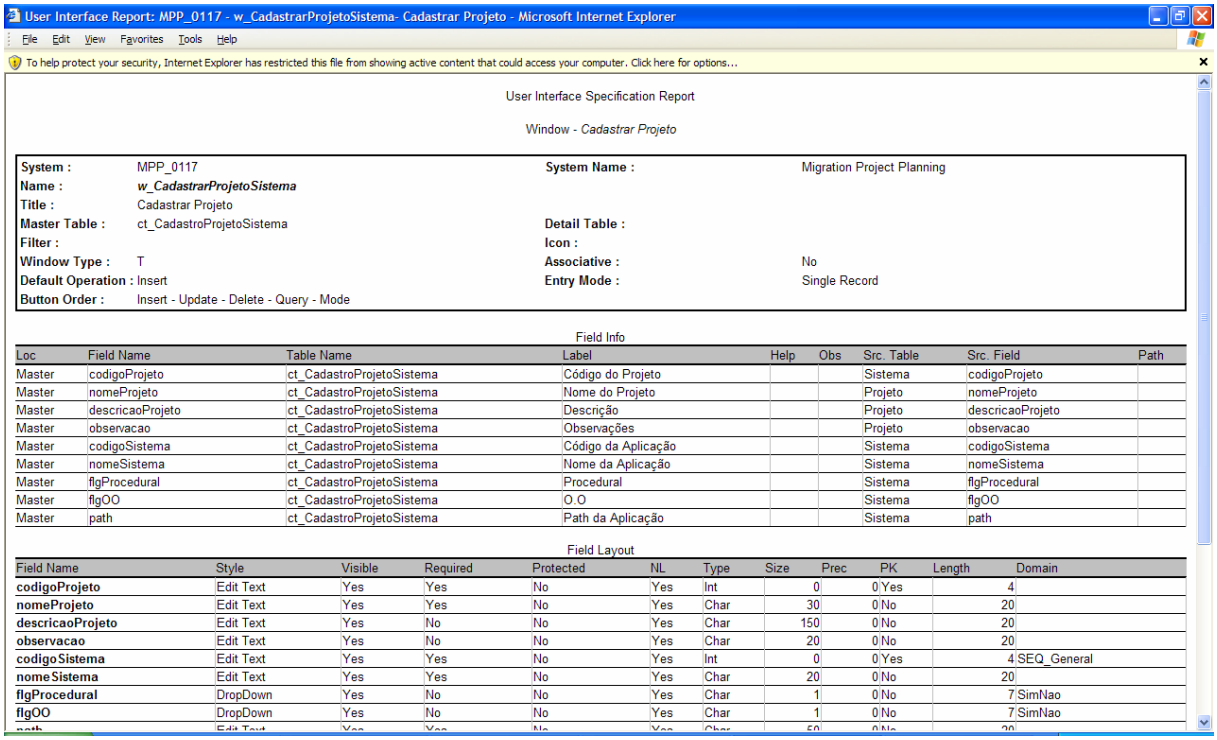
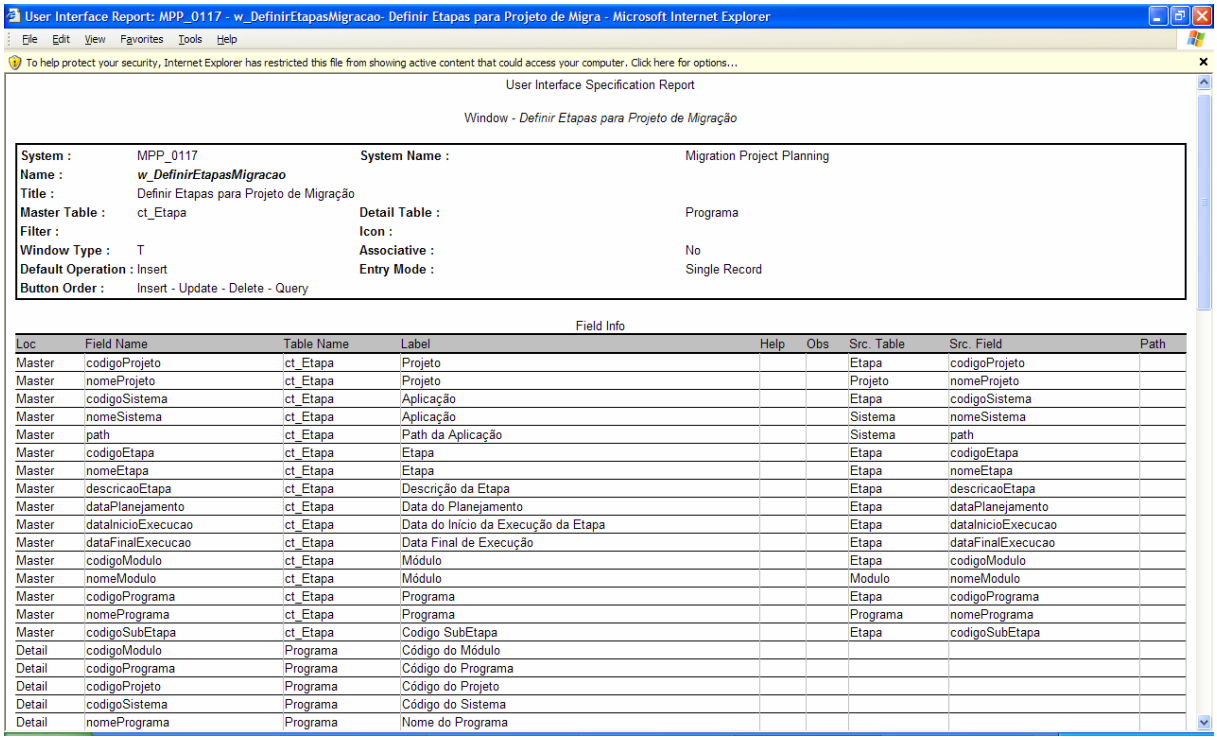


Figura 101 – Relatório de especificação da interface w_CadastrarProjetoSistema



User Interface Report: MPP_0117 - w_DefinirEtapasMigracao- Definir Etapas para Projeto de Migra - Microsoft Internet Explorer

File Edit View Favorites Tools Help

To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options...

User Interface Specification Report

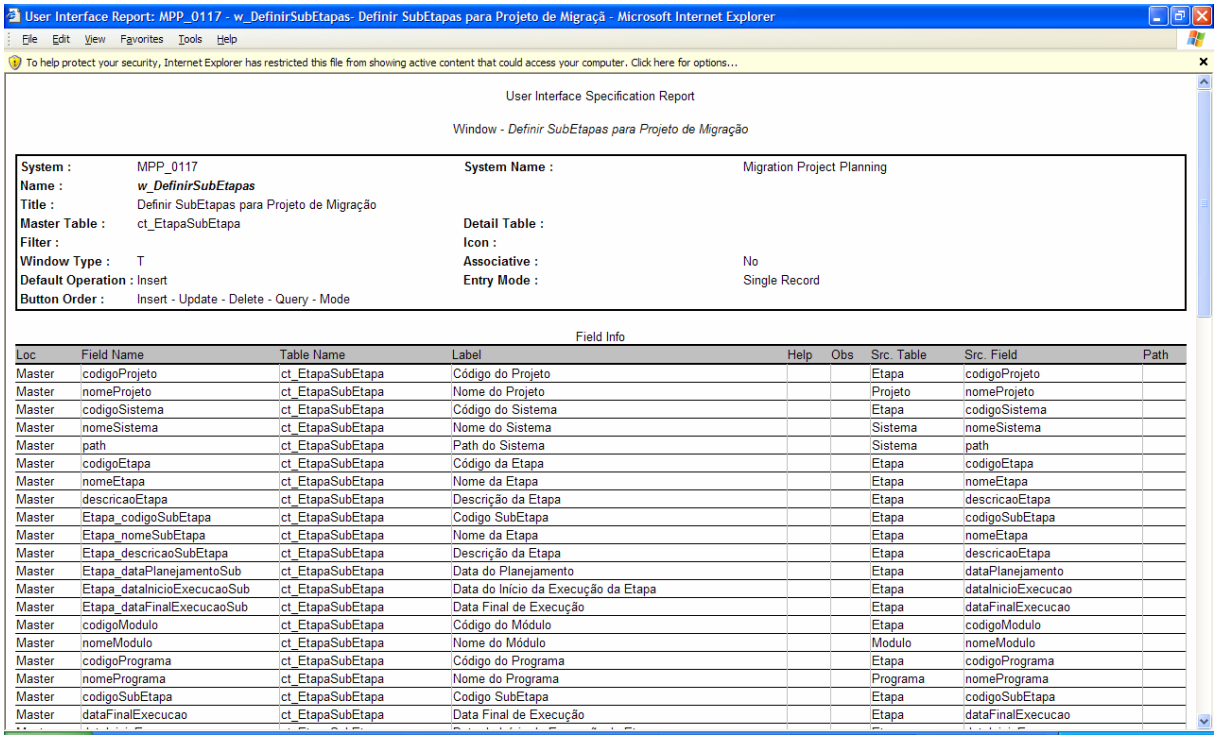
Window - *Definir Etapas para Projeto de Migração*

System :	MPP_0117	**System Name :**	Migration Project Planning
Name :	***w_DefinirEtapasMigracao***		
Title :	Definir Etapas para Projeto de Migração		
Master Table :	ct_Etapa	**Detail Table :**	Programa
Filter :		**Icon :**	
Window Type :	T	**Associative :**	No
Default Operation :	Insert	**Entry Mode :**	Single Record
Button Order :	Insert - Update - Delete - Query		

Field Info

Loc	Field Name	Table Name	Label	Help	Obs	Src. Table	Src. Field	Path
Master	codigoProjeto	ct_Etapa	Projeto			Etapa	codigoProjeto	
Master	nomeProjeto	ct_Etapa	Projeto			Projeto	nomeProjeto	
Master	codigoSistema	ct_Etapa	Aplicação			Etapa	codigoSistema	
Master	nomeSistema	ct_Etapa	Aplicação			Sistema	nomeSistema	
Master	path	ct_Etapa	Path da Aplicação			Sistema	path	
Master	codigoEtapa	ct_Etapa	Etapa			Etapa	codigoEtapa	
Master	nomeEtapa	ct_Etapa	Etapa			Etapa	nomeEtapa	
Master	descricaoEtapa	ct_Etapa	Descrição da Etapa			Etapa	descricaoEtapa	
Master	dataPlanejamento	ct_Etapa	Data do Planejamento			Etapa	dataPlanejamento	
Master	dataInicioExecucao	ct_Etapa	Data do Início da Execução da Etapa			Etapa	dataInicioExecucao	
Master	dataFinalExecucao	ct_Etapa	Data Final de Execução			Etapa	dataFinalExecucao	
Master	codigoModulo	ct_Etapa	Módulo			Etapa	codigoModulo	
Master	nomeModulo	ct_Etapa	Módulo			Modulo	nomeModulo	
Master	codigoPrograma	ct_Etapa	Programa			Etapa	codigoPrograma	
Master	nomePrograma	ct_Etapa	Programa			Programa	nomePrograma	
Master	codigoSubEtapa	ct_Etapa	Codigo SubEtapa			Etapa	codigoSubEtapa	
Detail	codigoModulo	Programa	Código do Módulo					
Detail	codigoPrograma	Programa	Código do Programa					
Detail	codigoProjeto	Programa	Código do Projeto					
Detail	codigoSistema	Programa	Código do Sistema					
Detail	nomePrograma	Programa	Nome do Programa					

Figura 102 – Relatório de especificação da interface w_DefinirEtapasMigracao



User Interface Report: MPP_0117 - w_DefinirSubEtapas- Definir SubEtapas para Projeto de Migraçã - Microsoft Internet Explorer

File Edit View Favorites Tools Help

To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options...

User Interface Specification Report

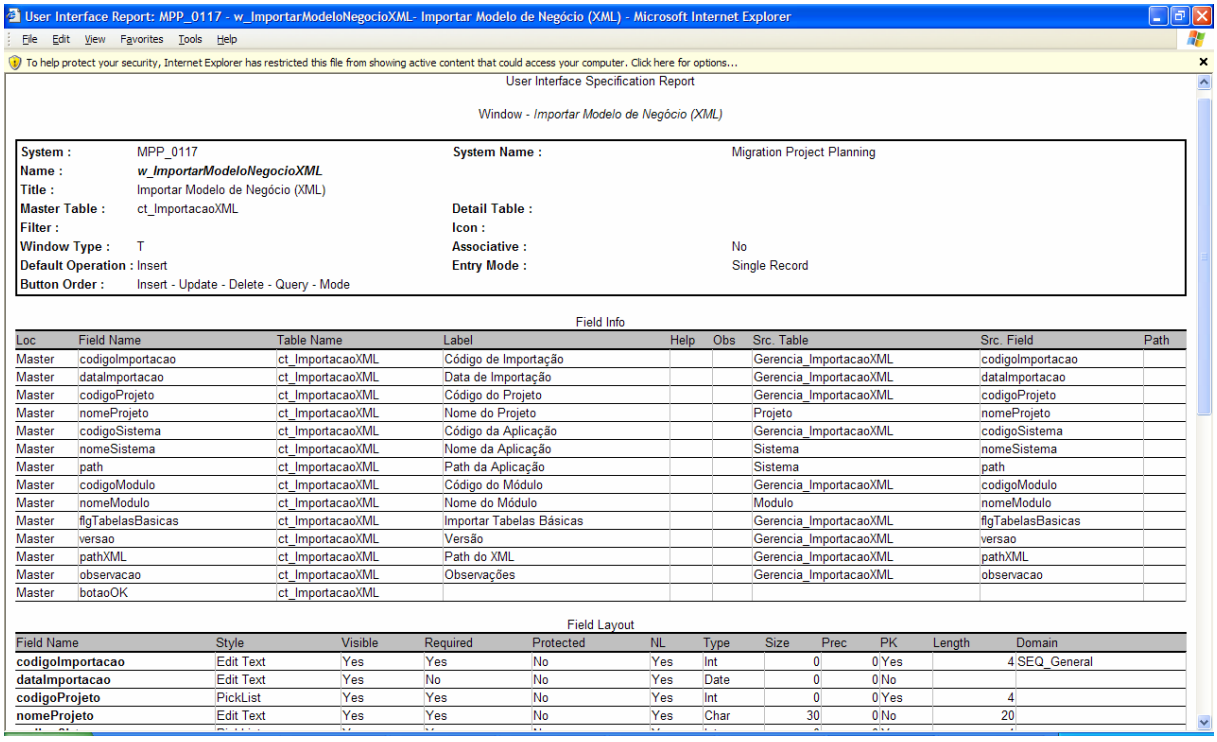
Window - *Definir SubEtapas para Projeto de Migração*

System :	MPP_0117	**System Name :**	Migration Project Planning
Name :	***w_DefinirSubEtapas***		
Title :	Definir SubEtapas para Projeto de Migração		
Master Table :	ct_EtapaSubEtapa	**Detail Table :**	
Filter :		**Icon :**	
Window Type :	T	**Associative :**	No
Default Operation :	Insert	**Entry Mode :**	Single Record
Button Order :	Insert - Update - Delete - Query - Mode		

Field Info

Loc	Field Name	Table Name	Label	Help	Obs	Src. Table	Src. Field	Path
Master	codigoProjeto	ct_EtapaSubEtapa	Código do Projeto			Etapa	codigoProjeto	
Master	nomeProjeto	ct_EtapaSubEtapa	Nome do Projeto			Projeto	nomeProjeto	
Master	codigoSistema	ct_EtapaSubEtapa	Código do Sistema			Etapa	codigoSistema	
Master	nomeSistema	ct_EtapaSubEtapa	Nome do Sistema			Sistema	nomeSistema	
Master	path	ct_EtapaSubEtapa	Path do Sistema			Sistema	path	
Master	codigoEtapa	ct_EtapaSubEtapa	Código da Etapa			Etapa	codigoEtapa	
Master	nomeEtapa	ct_EtapaSubEtapa	Nome da Etapa			Etapa	nomeEtapa	
Master	descricaoEtapa	ct_EtapaSubEtapa	Descrição da Etapa			Etapa	descricaoEtapa	
Master	Etapa_codigoSubEtapa	ct_EtapaSubEtapa	Codigo SubEtapa			Etapa	codigoSubEtapa	
Master	Etapa_nomeSubEtapa	ct_EtapaSubEtapa	Nome da Etapa			Etapa	nomeEtapa	
Master	Etapa_descricaoSubEtapa	ct_EtapaSubEtapa	Descrição da Etapa			Etapa	descricaoEtapa	
Master	Etapa_dataPlanejamentoSub	ct_EtapaSubEtapa	Data do Planejamento			Etapa	dataPlanejamento	
Master	Etapa_dataInicioExecucaoSub	ct_EtapaSubEtapa	Data do Início da Execução da Etapa			Etapa	dataInicioExecucao	
Master	Etapa_dataFinalExecucaoSub	ct_EtapaSubEtapa	Data Final de Execução			Etapa	dataFinalExecucao	
Master	codigoModulo	ct_EtapaSubEtapa	Código do Módulo			Etapa	codigoModulo	
Master	nomeModulo	ct_EtapaSubEtapa	Nome do Módulo			Modulo	nomeModulo	
Master	codigoPrograma	ct_EtapaSubEtapa	Código do Programa			Etapa	codigoPrograma	
Master	nomePrograma	ct_EtapaSubEtapa	Nome do Programa			Programa	nomePrograma	
Master	codigoSubEtapa	ct_EtapaSubEtapa	Codigo SubEtapa			Etapa	codigoSubEtapa	
Master	dataFinalExecucao	ct_EtapaSubEtapa	Data Final de Execução			Etapa	dataFinalExecucao	

Figura 103 – Relatório de especificação da interface w_DefinirSubEtapas



User Interface Report: MPP_0117 - w_ImportarModeloNegocioXML- Importar Modelo de Negócio (XML) - Microsoft Internet Explorer

File Edit View Favorites Tools Help

To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options...

User Interface Specification Report

Window - *Importar Modelo de Negócio (XML)*

System :	MPP_0117	**System Name :**	Migration Project Planning
Name :	***w_ImportarModeloNegocioXML***		
Title :	Importar Modelo de Negócio (XML)		
Master Table :	ct_ImportacaoXML	**Detail Table :**	
Filter :		**Icon :**	
Window Type :	T	**Associative :**	No
Default Operation :	Insert	**Entry Mode :**	Single Record
Button Order :	Insert - Update - Delete - Query - Mode		

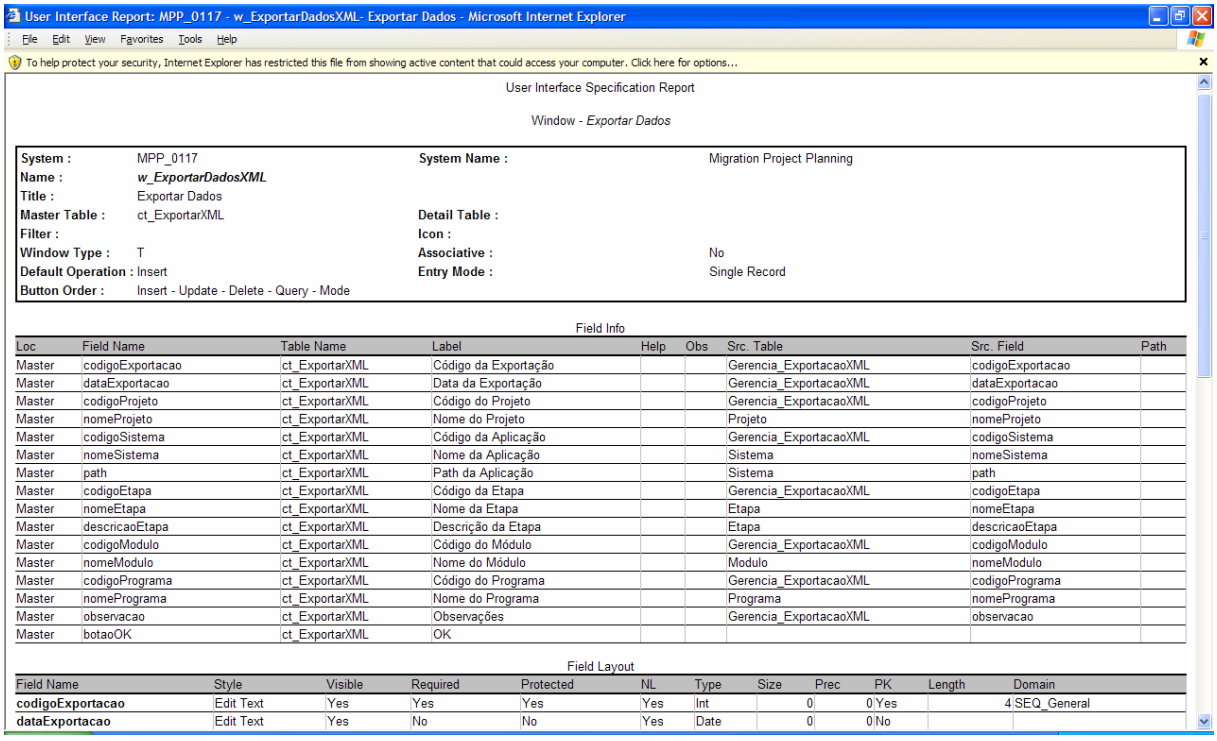
Field Info

Loc	Field Name	Table Name	Label	Help	Obs	Src. Table	Src. Field	Path
Master	codigoImportacao	ct_ImportacaoXML	Código de Importação			Gerencia_ImportacaoXML	codigoImportacao	
Master	dataImportacao	ct_ImportacaoXML	Data de Importação			Gerencia_ImportacaoXML	dataImportacao	
Master	codigoProjeto	ct_ImportacaoXML	Código do Projeto			Gerencia_ImportacaoXML	codigoProjeto	
Master	nomeProjeto	ct_ImportacaoXML	Nome do Projeto			Projeto	nomeProjeto	
Master	codigoSistema	ct_ImportacaoXML	Código da Aplicação			Gerencia_ImportacaoXML	codigoSistema	
Master	nomeSistema	ct_ImportacaoXML	Nome da Aplicação			Sistema	nomeSistema	
Master	path	ct_ImportacaoXML	Path da Aplicação			Sistema	path	
Master	codigoModulo	ct_ImportacaoXML	Código do Módulo			Gerencia_ImportacaoXML	codigoModulo	
Master	nomeModulo	ct_ImportacaoXML	Nome do Módulo			Modulo	nomeModulo	
Master	flgTabelasBasicas	ct_ImportacaoXML	Importar Tabelas Básicas			Gerencia_ImportacaoXML	flgTabelasBasicas	
Master	versao	ct_ImportacaoXML	Versão			Gerencia_ImportacaoXML	versao	
Master	pathXML	ct_ImportacaoXML	Path do XML			Gerencia_ImportacaoXML	pathXML	
Master	observacao	ct_ImportacaoXML	Observações			Gerencia_ImportacaoXML	observacao	
Master	botaoOK	ct_ImportacaoXML						

Field Layout

Field Name	Style	Visible	Required	Protected	NL	Type	Size	Prec	PK	Length	Domain
codigoImportacao	Edit Text	Yes	Yes	No	Yes	Int	0	0	Yes	4	SEQ_General
dataImportacao	Edit Text	Yes	No	No	Yes	Date	0	0	No		
codigoProjeto	PickList	Yes	Yes	No	Yes	Int	0	0	Yes	4	
nomeProjeto	Edit Text	Yes	Yes	No	Yes	Char	30	0	No	20	

Figura 104 – Relatório de especificação da interface w_ImportarModeloNegocioXML



User Interface Report: MPP_0117 - w_ExportarDadosXML- Exportar Dados - Microsoft Internet Explorer

File Edit View Favorites Tools Help

To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options...

User Interface Specification Report

Window - *Exportar Dados*

System :	MPP_0117	**System Name :**	Migration Project Planning
Name :	***w_ExportarDadosXML***		
Title :	Exportar Dados		
Master Table :	ct_ExportarXML	**Detail Table :**	
Filter :		**Icon :**	
Window Type :	T	**Associative :**	No
Default Operation :	Insert	**Entry Mode :**	Single Record
Button Order :	Insert - Update - Delete - Query - Mode		

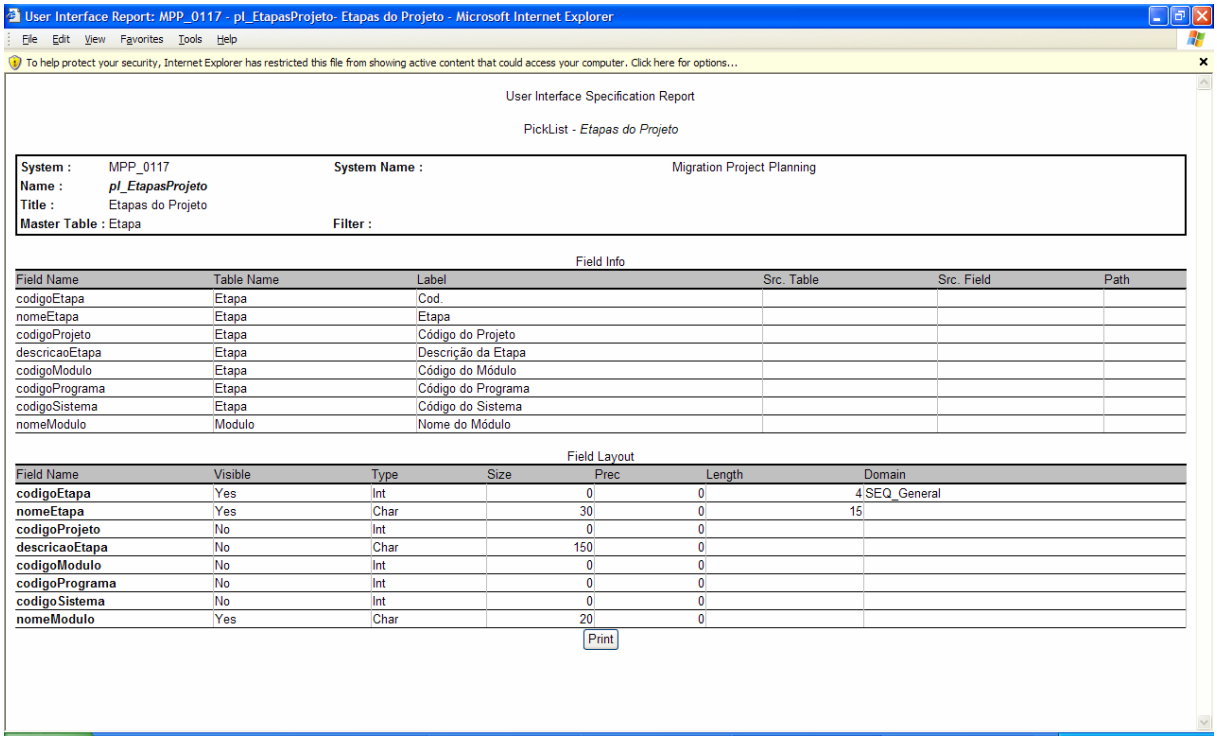
Field Info

Loc	Field Name	Table Name	Label	Help	Obs	Src. Table	Src. Field	Path
Master	codigoExportacao	ct_ExportarXML	Código da Exportação			Gerencia_ExportacaoXML	codigoExportacao	
Master	dataExportacao	ct_ExportarXML	Data da Exportação			Gerencia_ExportacaoXML	dataExportacao	
Master	codigoProjeto	ct_ExportarXML	Código do Projeto			Gerencia_ExportacaoXML	codigoProjeto	
Master	nomeProjeto	ct_ExportarXML	Nome do Projeto			Projeto	nomeProjeto	
Master	codigoSistema	ct_ExportarXML	Código da Aplicação			Gerencia_ExportacaoXML	codigoSistema	
Master	nomeSistema	ct_ExportarXML	Nome da Aplicação			Sistema	nomeSistema	
Master	path	ct_ExportarXML	Path da Aplicação			Sistema	path	
Master	codigoEtapa	ct_ExportarXML	Código da Etapa			Gerencia_ExportacaoXML	codigoEtapa	
Master	nomeEtapa	ct_ExportarXML	Nome da Etapa			Etapa	nomeEtapa	
Master	descricaoEtapa	ct_ExportarXML	Descrição da Etapa			Etapa	descricaoEtapa	
Master	codigoModulo	ct_ExportarXML	Código do Módulo			Gerencia_ExportacaoXML	codigoModulo	
Master	nomeModulo	ct_ExportarXML	Nome do Módulo			Modulo	nomeModulo	
Master	codigoPrograma	ct_ExportarXML	Código do Programa			Gerencia_ExportacaoXML	codigoPrograma	
Master	nomePrograma	ct_ExportarXML	Nome do Programa			Programa	nomePrograma	
Master	observacao	ct_ExportarXML	Observações			Gerencia_ExportacaoXML	observacao	
Master	botaoOK	ct_ExportarXML	OK					

Field Layout

Field Name	Style	Visible	Required	Protected	NL	Type	Size	Prec	PK	Length	Domain
codigoExportacao	Edit Text	Yes	Yes	Yes	Yes	Int	0	0	Yes	4	SEQ_General
dataExportacao	Edit Text	Yes	No	No	Yes	Date	0	0	No		

Figura 105 – Relatório de especificação da interface w_ExportarDadosXML



User Interface Report: MPP_0117 - pl_EtapasProjeto- Etapas do Projeto - Microsoft Internet Explorer

File Edit View Favorites Tools Help

To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options...

User Interface Specification Report

PickList - *Etapas do Projeto*

System :	MPP_0117	**System Name :**	Migration Project Planning
Name :	***pl_EtapasProjeto***		
Title :	Etapas do Projeto		
Master Table :	Etapa	**Filter :**	

Field Info

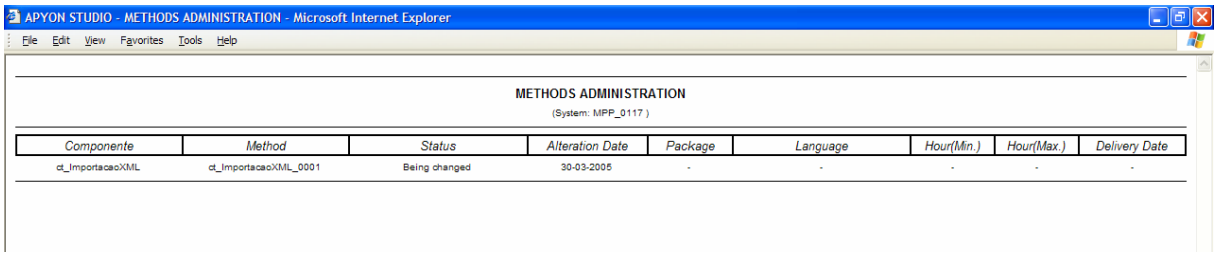
Field Name	Table Name	Label	Src. Table	Src. Field	Path
codigoEtapa	Etapa	Cod.			
nomeEtapa	Etapa	Etapa			
codigoProjeto	Etapa	Código do Projeto			
descricaoEtapa	Etapa	Descrição da Etapa			
codigoModulo	Etapa	Código do Módulo			
codigoPrograma	Etapa	Código do Programa			
codigoSistema	Etapa	Código do Sistema			
nomeModulo	Modulo	Nome do Módulo			

Field Layout

Field Name	Visible	Type	Size	Prec	Length	Domain
codigoEtapa	Yes	Int	0	0	4	SEQ_General
nomeEtapa	Yes	Char	30	0	15	
codigoProjeto	No	Int	0	0		
descricaoEtapa	No	Char	150	0		
codigoModulo	No	Int	0	0		
codigoPrograma	No	Int	0	0		
codigoSistema	No	Int	0	0		
nomeModulo	Yes	Char	20	0		

Print

Figura 106 – Relatório de especificação da PickList pl_EtapasProjeto



APYON STUDIO - METHODS ADMINISTRATION - Microsoft Internet Explorer

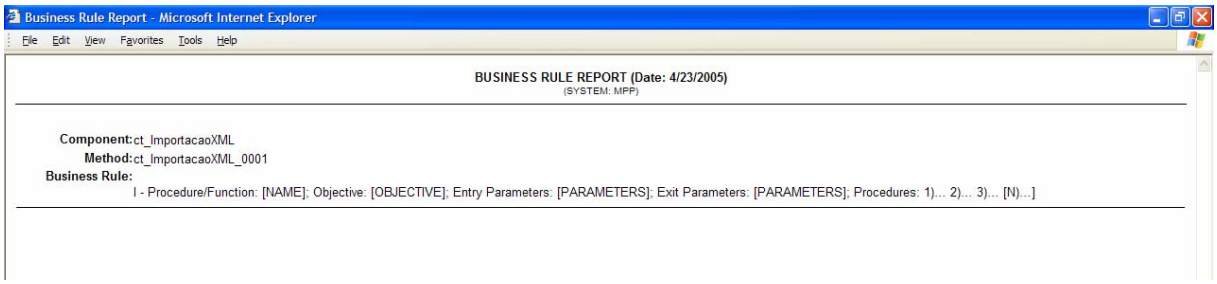
File Edit View Favorites Tools Help

METHODS ADMINISTRATION

(System: MPP_0117)

Componente	*Method*	*Status*	*Alteration Date*	*Package*	*Language*	*Hour(Min.)*	*Hour(Max.)*	*Delivery Date*
ct_ImportacaoXML	ct_ImportacaoXML_0001	Being changed	30-03-2005	-	-	-	-	-

Figura 107 – Relatório de especificação do método ct_ImportacaoXML_0001



Business Rule Report - Microsoft Internet Explorer

File Edit View Favorites Tools Help

BUSINESS RULE REPORT (Date: 4/23/2005)

(SYSTEM: MPP)

Component: ct_ImportacaoXML

Method: ct_ImportacaoXML_0001

Business Rule:

I - Procedure/Function: [NAME]; Objective: [OBJECTIVE]; Entry Parameters: [PARAMETERS]; Exit Parameters: [PARAMETERS]; Procedures: 1)... 2)... 3)... [N)...]

Figura 108 – Relatório de especificação da regra de negócio do método ct_ImportacaoXML_0001

entrada
↓
Regra de
transforma
ção
'

O tempo passa, traz e leva; leva a dor, as mágoas, mas leva também momentos que desejaríamos viver eternamente. Entretanto, ele nos traz novas experiências, ansiedades e receios, sentimentos que tornam nossa vida repleta de emoções. Isso muitas vezes assusta, amedronta, mas Deus, em sua extrema bondade, não nos deixa desanimar. Antes, Ele nos concede forças para batalharmos, irmos avante em nossa longa caminhada, perseguindo os nossos sonhos, nossos anseios, com a esperança de ao fim sermos vitoriosos.

Fabiano Pinatti