TB-REPP - Padrões de Processo para a Engenharia Reversa baseado em Transformações [1]

Darley Rosa Peres[1]
Alexandre Alvaro[1]
Valdirene Fontanette[1]
Vinicius Cardoso Garcia[1][3]
Antonio Francisco do Prado[1]
Rosana Teresinha Vaccare Braga[2]

[1]Departamento de Computação – Universidade Federal de São Carlos
Rod. Washington Luiz, km 235 – São Carlos/SP - Brazil
Caixa Postal 676 – Cep 13565-905 Fone/Fax: + 55-16-260-8232
{darley, aalvaro, valdirene, vinicius, prado}@dc.ufscar.br

[2]Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – USP - São Carlos
rtvb@icmc.usp.br

Abstract. Reverse Engineering is a technique used to recover information from software documents and even from source code, aiming to obtain its representation in a higher abstraction level. In that way, it can make easier the system understanding.
Considering that in the Re-engineering process, Reverse Engineering is the most problematic phase, this paper proposes a Transformation-Based Reverse Engineering Process Patterns group (TB-REPP), to obtain the system project in a high abstraction level, assuring its evolution and maintenance, making it more expressive and easy to understand.

1. Introdução

Sistema de software é um artefato evolutivo e requer constantes modificações, seja para corrigir erros, melhorar desempenho, adicionar novas funcionalidades ou até mesmo para adaptá-lo às novas tecnologias de desenvolvimento de software.

A dificuldade em atualizar esses sistemas de software tem motivado pesquisadores a encontrar novas soluções para facilitar e reduzir os custos de sua manutenção, pois a falta de uma documentação consistente que permita um melhor entendimento do domínio da aplicação é constante na grande maioria dos sistemas.

Atualmente, existe um grande número de empresas que continuam trabalhando com sistemas implementados em linguagens de programação antigas já em desuso, cujas manutenções são árduas e custosas. Esses sistemas, denominados legados, ainda são de muita utilidade aos seus usuários e muitas vezes, sua reconstrução, usando técnicas modernas de desenvolvimento de software, podem ser a solução para sua reutilização sem ter que construir um novo sistema. Normalmente, os sistemas legados não possuem documentação ou quando possuem, a mesma está desatualizada, devido às inúmeras alterações e adaptações neles implementadas, e nem sempre a pessoa que desenvolveu o sistema é a responsável pela sua manutenção. Fatores como estes tornam difícil e de alto custo a manutenção, fazendo com que os desenvolvedores até abandonem o sistema.

Num processo de Reengenharia, a fase mais árdua e custosa fica a cargo da Engenharia Reversa, pois a obtenção de uma documentação e da recuperação de um projeto orientado a objetos partindo de um código legado é mais difícil do que aplicar a Engenharia Avante com o projeto já recuperado e todo documentado. Levando em consideração essa dificuldade, é proposto um conjunto de Padrões de Processo para a Engenharia Reversa

[3] Financiado pela Fundação de Amparo à Pesquisa do Estado da Bahia

baseada em Transformações, cuja contribuição é a obtenção de um projeto recuperado do sistema legado em um alto nível de abstração, representado com técnicas *UML* [06,07], garantindo sua evolução e sua manutenibilidade, tornando-o mais expressivo e de fácil entendimento. Esses Padrões utilizam-se de transformações que além de agilizar o processo de Engenharia Reversa, facilitam a obtenção da documentação do projeto.

Transformação de software consiste em gerar novas descrições dos programas numa mesma linguagem ou em outras linguagens. Pode ser usado tanto para implementar um sistema, a partir de especificações em alto nível de abstração, quanto para reimplementar um software existente a partir de seu código fonte, com o objetivo de atualizá-lo para ser executado em outras plataformas computacionais. Pode compreender não só a mudança de código, mas também uma mudança de paradigmas de programação, interfaces de usuário e arquiteturas de banco de dados.

Os padrões existentes em cada passo do processo são apresentados na Tabela 1, juntamente com suas descrições.

Nome Padrão	**Descrição**
Passo Identificar	
1. Definir hierarquia Chama/Chamado	Identificar no código legado a hierarquia de chamadas das unidades de programas.
2. Identificar Casos de Uso	Identificar casos de uso do código legado, e armazená-los como fatos na base de conhecimento.
3. Identificar Supostas Classes e Supostos Atributos	Identificar no código legado, supostas classes e supostos atributos que darão origem a classes e atributos no projeto final.
4. Identificar Supostos Métodos	Identificar fatos sobre supostos métodos e armazená-los na base de conhecimento.
5. Identificar Relacionamentos	Identificar os supostos relacionamentos e seus tipos e armazená-los na base de conhecimentos
6. Identificar Cardinalidades	Identificar as cardinalidades dos relacionamentos e armazená-las como fatos na base de conhecimento.
Passo Organizar	
1. Criar Supostas Classes e Supostos Atributos	Através dos fatos armazenados na base de conhecimento criar arquivos contendo as supostas classes e supostos atributos.
2. Alocar Supostos Métodos nas Supostas Classes	Alocar os supostos métodos que estão armazenados como fatos na base de conhecimento em suas supostas classes.
Passo Recuperar	
1. Criar Especificações das Classes e dos Atributos	Criar as especificações na linguagem de modelagem para as classes e seus atributos.
2. Criar Especificações dos Métodos	Criar as especificações dos métodos na linguagem de modelagem em suas respectivas classes.
3. Criar Casos de Uso	Gerar as especificações dos Casos de Uso na linguagem de modelagem.
4. Criar Diagrama de Seqüência	Criar as especificações em linguagem de modelagem para o diagrama de seqüência, seguindo o fluxo de controle do código legado organizado e dos casos de uso.
5. Criar Especificações dos Relacionamentos e Cardinalidades	Gerar as especificações dos relacionamentos e suas cardinalidades na linguagem de modelagem, de acordo com os fatos identificados.

Tabela 1. Catálogo dos Padrões de Processos.

Passo Identificar

O objetivo deste passo é identificar no código legado os elementos que compõem o pseudo Modelo Orientado a Objetos (OO). O Engenheiro de Software, utilizando-se de transformações de software que extrai essas informações sobre a estrutura do código legado, identificando de forma semi-automática elementos tais como, supostas classes, seus supostos atributos, supostos métodos, supostos relacionamentos, supostos casos de uso e a hierarquia de chamadas do código legado, e armazena-os como fatos numa Base de Conhecimento. Esses fatos auxiliarão na organização do código, segundo os princípios da Orientação a Objetos.

Para facilitar a identificação desses elementos foi adotado um *"Metamodelo"*, segundo a notação *UML* [08], que representa os modelos Orientados a Objetos de um sistema, conforme mostra a Figura 1.

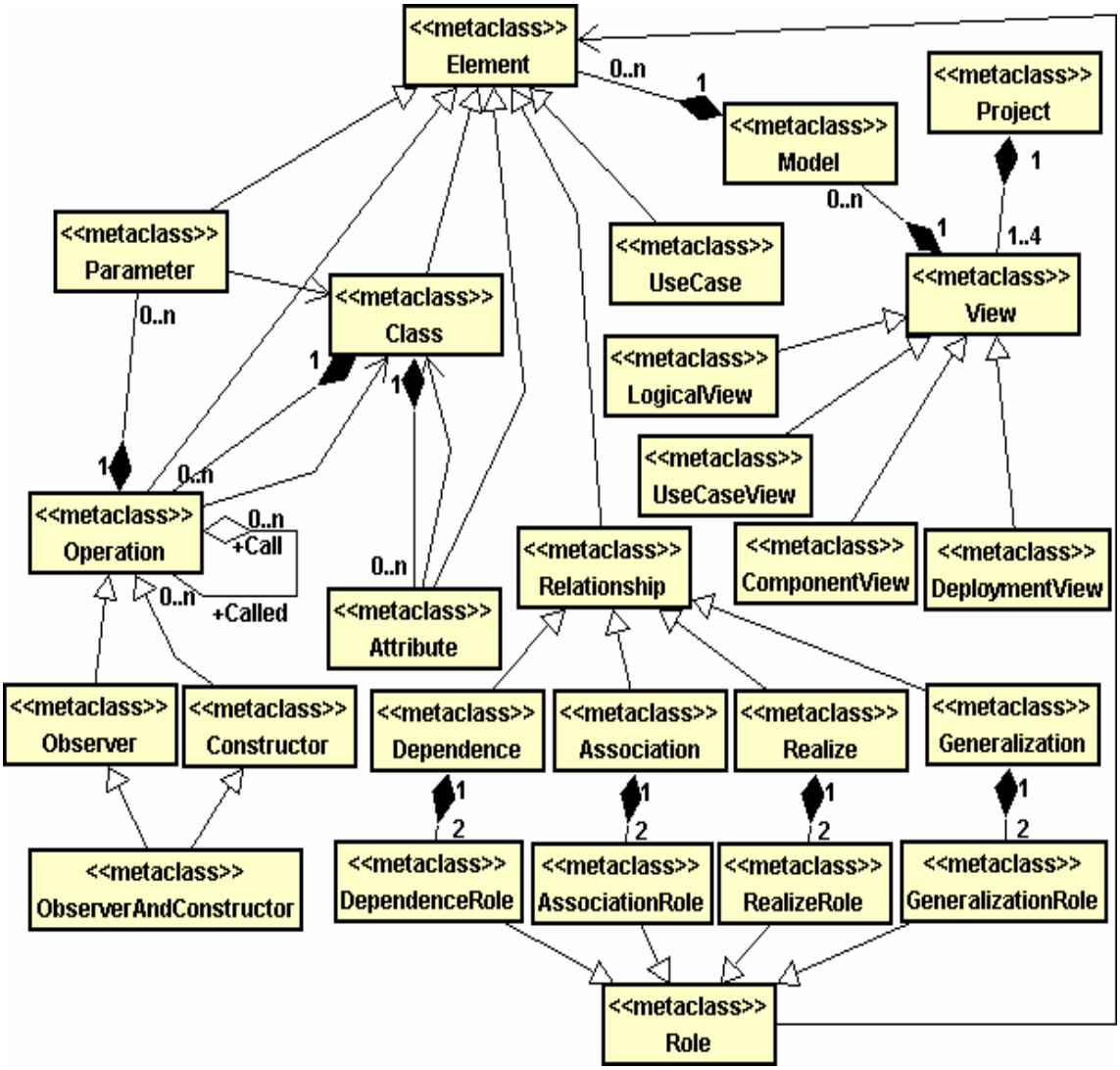


Figura 1. Metamodelo das especificações Orientadas a Objetos na notação UML

1. Definir Hierarquia de Chamadas (CHAMA/CHAMADO)

Intuito:

- Determinar o fluxo de execução das unidades de programa do sistema legado.

Problema:

- Há uma necessidade de se conhecer a hierarquia dos programas/módulos do sistema legado para identificar a seqüência de execução dessas unidades de programas.

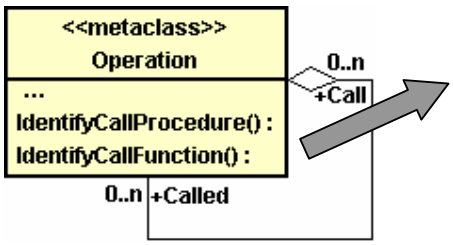
Influências:

- Ausência de documentação, exceto o código fonte (programas).
- A existência de grande quantidade de arquivos de programas e de dados nos sistemas legados, dificultando o seu entendimento.

Solução:

Através de transformações, com o auxílio do Engenheiro de Software, obter uma relação de unidades de programas (programas, procedimentos e funções) CHAMA/CHAMADO [09], fazendo uma referência cruzada entre essas unidades de programas, ou seja, identificando as unidades de programas por elas utilizadas (chamadas) e as unidades de programas que as utilizam (chamadoras).

A Figura 2 mostra o auto-relacionamento "*Call Called"* da metaclasse "*Operation"*, utilizados na geração dos fatos sobre a hierarquia de chamadas. A figura mostra também a transformação para geração dos fatos.




```
TRANSFORM  IdentifyCallProcedure
LHS: {{dast domain.procedureRole
 ProcedureCallProgram([[name_TargetProgram]]);
}}
POST-MATCH: {{dast txt.decls
...
}}
TEMPLATE T_ hierarchy
RHS: {{dast txt. decls
          [decls_CallerProgram]] call
          [[decls_TargetProgram]]
}}
```

Figura 2. Metaclasse *CallCalledHierarchy* do Metamodelo *OO*

A Figura 3 mostra um exemplo que ilustra a relação CHAMA/CHAMADO, onde as informações obtidas são baseadas na metaclasse "*Operation"* do metamodelo OO.

Na relação CHAMA/CHAMADO, se um procedimento ou função é chamado, registra-se também a unidade de programa em que ele se encontra. Por exemplo, a relação CHAMA/CHAMADO da Figura 3 mostra que, em uma unidade de programa Clipper chamado *"*Catproc.prg*"* tem-se os procedimentos "Inicial*", "*Logotipo*"* e *"*Janela*"*, que estão sendo chamados por uma outra unidade de programa chamada *"*Menuprin.prg*"*. Essas informações são importantes, pois ajudam a identificar uma seqüência dos programas a serem submetidos aos transformadores. Neste caso, a ordem para submeter os programas aos transformadores será do "Catproc.prg" e por último o "Menuprin.prg".

CHAMA	CHAMADO			
Menuprin	Inicial (p)	<Catproc.prg>	Modos2 (prg)	
	Logotipo (p)	<Catproc.prg>	Elimios (prg)	
	Janela (p)	<Catproc.prg>	Insecve (prg)	
	Ostela1 (prg)		Altcve (prg)	
	Ostela2 (prg)		Elicve (prg)	
	Modos1 (prg)		Indexa (prg)	
Ostela1	Logotipo (p)	<Catproc.prg>	Impos1 (prg)	<Catproc.prg>
	Aviso (p)	<Catproc.prg>	Buscli (f)	<Catproc.prg>
	Telalim (p)	<Catproc.prg>	Coluna (f)	<Ostela1.prg>
	Limpa (p)	<Catproc.prg>	Busvei (f)	<Modos1.prg>

Figura 3. Relação CHAMA/CHAMADO

Padrões Relacionados:

- Este padrão é a entrada para os padrões Identificar Casos de Uso, utilizado neste passo e para os padrões Criar Casos de Uso e Criar Diagrama de Seqüência, utilizados no passo Recuperar.
- Relaciona-se com o padrão ***Iniciar Análise dos Dados*** proposto por [14], que propõe a construção de uma tabela de Programas x Arquivos com objetivo de se saber o relacionamento entre eles. Já na relação CHAMA/CHAMADO o relacionamento é entre os módulos para se obter o fluxo de execução do sistema.

2. Identificar Casos de Uso

Intuito:

- Identificar os casos de uso do sistema legado e armazená-los na base de conhecimento.

Problema:

- Os Casos de Uso deverão ser identificados através da inspeção do código legado, que muitas vezes é implementado em uma linguagem procedural e não Orientada a Objetos.

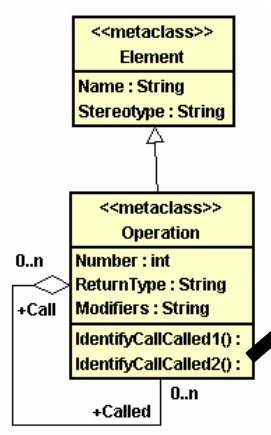
Influências:

- A falta de documentação para a identificação dos atores e Casos de Uso.
- A estrutura do código de sistemas legados procedimentais.

Solução:

Baseando-se na hierarquia de chamadas, e com o auxílio do Engenheiro de Software, aplicar transformações que deverão identificar e armazenar na base de conhecimento, fatos sobre os atores, que representam qualquer entidade que interage com o sistema, os casos de uso e a interação entre eles. Em seguida, através da interação do Engenheiro de Software, que deverá apontar o início de um caso de uso como sendo a chamada que inicia a execução de um determinado cenário significativamente importante para o sistema, como por exemplo, a chamada de uma unidade de programa do menu principal do sistema.

A Figura 4 exemplifica a transformação de identificação dos fatos dos supostos Casos de Uso, Atores e suas interações, que serão armazenadas na base de conhecimento.




```
TRANSFORM IdentifyCallCalled2
LHS: {{dast clipper.call_func_proc
   'DO' [[iden chamado]]
}}

POST-MATCH: {{dast txt.decls
...
fprintf(stderr," ** USE CASE IDENTIFICADO **");
fprintf(stderr,"O ATOR deve ser informado !");
cin >> ator;
fprintf(stderr,"O USE CASE deve ser informado !");
cin >> usecase;
...
}
   KBAssertIfNew("usecase_diagram([[ator]], [[usecase]])");
...
}}
```

Figura 4. Transformação para a identificação de supostos Casos de Uso

O resultado disso é o particionamento da hierarquia de chamadas em Casos de Uso.

Padrões Relacionados:

- Esse padrão tem como entrada o padrão Definir Hierarquia de Chama/Chamado, e é utilizado como entrada no padrão Criar Casos de Uso no passo Recuperar.
- Relaciona-se com o padrão ***"Obter Cenário"*** proposto por [14], cujo objetivo é obter os cenários do sistema através da análise das interfaces do sistema em operação, gerando a tabela de cenários do sistema.

3. Identificar Supostas Classes e Supostos Atributos

Intuito:

- Identificar no código legado, supostas classes e supostos atributos e armazená-los como fatos na Base de Conhecimento.

Problema:

- Diferentes formas de acesso a Banco de Dados e arquivos nos diferentes domínios do código legado e falta de uma estruturação dos dados.
- A desorganização do código legado dificulta bastante o trabalho de identificação de classes e atributos.

Influências:

- Reconhecer classes/objetos e seus relacionamentos a partir do código legado é difícil.
- Utilizar transformações para reconhecer fatos que poderão dar origem às respectivas classes e atributos.

Solução:

Utilizando-se de transformações de identificação, identificar as supostas classes e seus supostos atributos considerando-se os comandos de abertura e acesso aos arquivos ou banco de dados, comandos relacionados com as estruturas de dados, globais ou locais e as interfaces de interação do sistema com o ambiente externo.

O Engenheiro de Software deverá considerar as seguintes possibilidades:

- Sistemas que utilizam arquivos de dados:
  - ***Supostas Classes***
    Caso o sistema utilize arquivos de dados, cada arquivo de dados dá origem a uma suposta classe persistente da camada Banco de dados e cada estrutura de dados dá origem a uma suposta classe da camada Regras de Negócio. As telas de interação com o ambiente externo dão origem a supostas classes da camada de Interface.
  - ***Supostos Atributos***
    Caso o sistema utilize arquivos de dados, cada campo do arquivo de dados dá origem a um suposto atributo desse arquivo de dados.

- Sistemas que utilizam banco de dados:
  - ***Supostas Classes***
    Caso o sistema utilize banco de dados, cada tabela do banco de dados dá origem a uma suposta classe persistente da camada Banco de dados e cada estrutura de dados dá origem a uma suposta classe da camada Regras de Negócio. As telas de interação com o ambiente externo dão origem a supostas classes da camada de Interface.
  - ***Supostos Atributos***
    Caso o sistema utilize banco de dados, cada campo da tabela do banco de dados dá origem a um suposto atributo dessa tabela.

As supostas classes e supostos atributos identificados pelas transformações são armazenados como fatos na Base de Conhecimento pelas próprias transformações. A estrutura destes fatos deverá ser organizada de acordo com as propriedades que definem um atributo. Essas propriedades são encontradas em forma de meta-atributos das metaclasses "*Class"* e *"Attribute"*, do metamodelo OO, como mostra a Figura 5. É ilustrados também um exemplo de transformação implementada para geração de fatos das supostas classes e supostos atributos e o armazenamento dos mesmos na Base de Conhecimento.

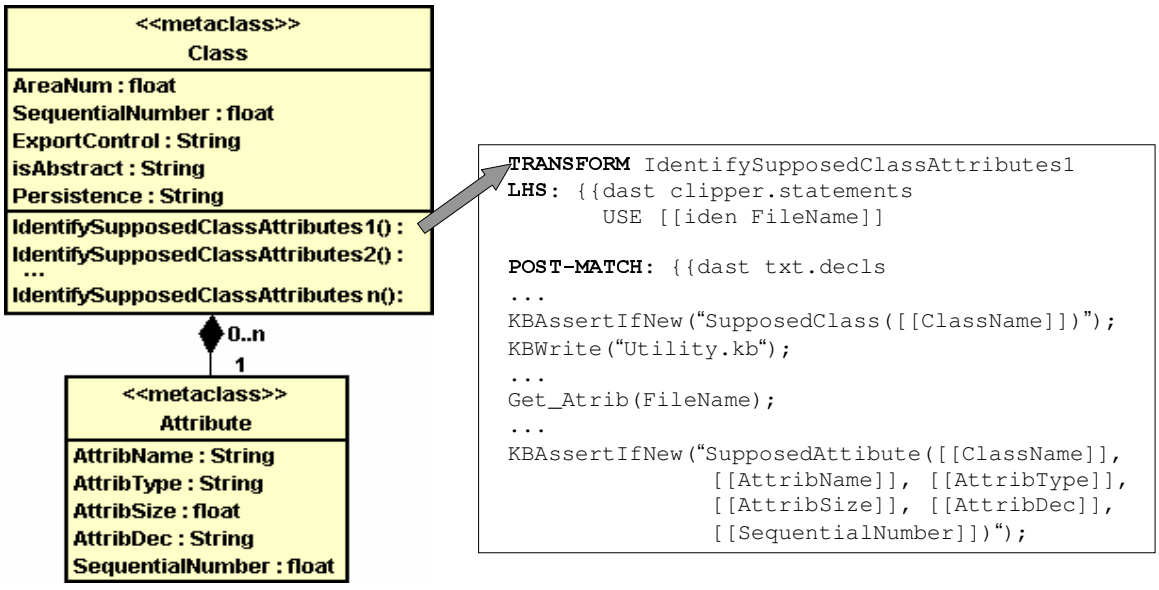


Figura 5. Metaclasse *Class* e *Attribute* do Metamodelo *OO*

Os meta-atributos *"Name", "Stereotype", "AttribName", "AttribType", "AttribSize", "AttribDec" e "SequentialNumber"* foram identificados na metaclasse *"Attribute"*. Deve-se verificar a existência de estruturas correspondentes para cada um desses meta-atributos no paradigma do código legado. Caso exista, a estrutura do fato do suposto atributo deverá conter esse dado.

Além disso, é criada uma suposta classe Principal cujo objetivo é alocar trechos que não podem ser alocados nas supostas classes de Negócio, como por exemplo, rotinas que iniciam o sistema.

Para ambos os casos, utilizando arquivos de dados ou banco de dados, as variáveis globais encontradas no código dão origem a atributos privados da suposta classe Principal.

Padrões Relacionados:

- Este padrão é a entrada para o padrão Criar Supostas Classes e Atributos, utilizado no passo Organizar.
- Relaciona-se com o padrão ***"Criar Visão OO dos Dados"*** proposto por [14], que tem como objetivo criar uma visão orientada a objetos dos dados, obtendo um Diagrama de Pseudo-Classes.

4. Identificar Supostos Métodos

Intuito:

- Identificar supostos métodos e armazená-los na base de conhecimento.

Problema:

- A dificuldade em identificar em sistemas legados, seus procedimentos, funções ou blocos de comandos.
- Anomalias podem ocorrer com comandos que consultam e/ou comandos que alteram a estrutura de dados.

Influências:

- Na maioria dos sistemas legados a estruturação do código não é modular, os códigos estão embaralhados em um procedimento principal, tornando quase impossível a identificação dos métodos.

Solução:

Identificar no código legado e armazenar na base de conhecimento, por meio das transformações, os supostos métodos a partir das unidades de programas que podem ser procedimentos, funções ou blocos de comandos executados pelo disparo de um evento de interface ou pela chamada de outra unidade de programa, respeitando o escopo, dependência de dados e visibilidade do código legado. Os supostos métodos são classificados em [09]:

- construtores (c), quando alteram uma estrutura de dados, e
- observadores (o), quando somente consultam a estrutura de dados.

Um suposto método que faz referência à somente um arquivo de dados é relacionado como suposto método da suposta classe identificada para esse arquivo. Se o suposto método não consulta nem modifica nenhum arquivo de dados, ele é relacionado com uma suposta classe de interface.

Para a identificação de um suposto método, as transformações deverão conter formas para verificar todas as características que um método necessita para ser representado na

notação UML, de acordo com o metamodelo. Essas características são encontradas em forma de atributos da metaclasse *"Operation"*, conforme mostra a Figura 6.

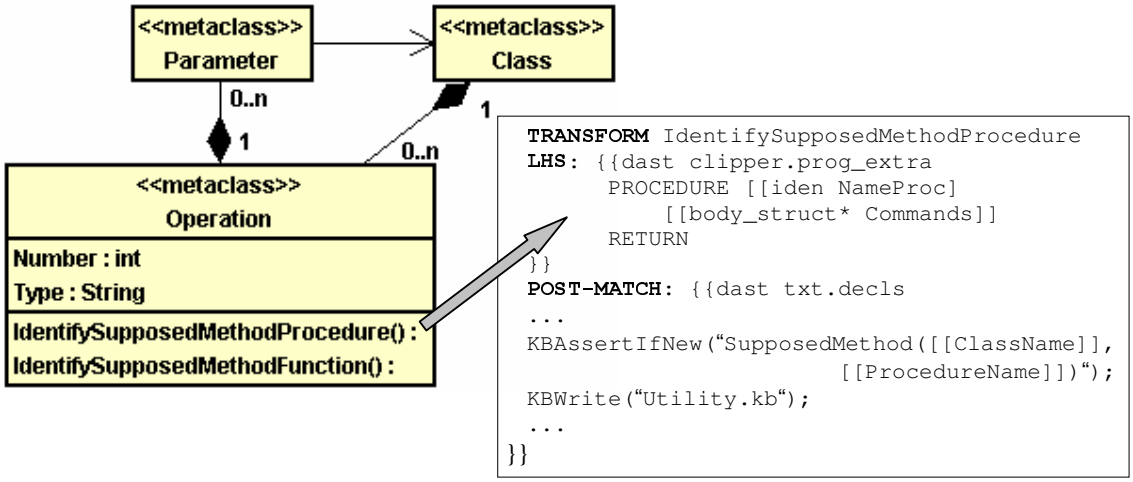


Figura 6. Metaclasse *Operation* do Metamodelo *OO*

Padrões Relacionados:

- Este padrão é a entrada para o padrão Alocar Supostos Métodos nas Supostas Classes, utilizado no passo Organizar.
- Relaciona-se com o padrão ***"Tratar Anomalias"*** proposto por [14], que tem como objetivo analisar as Descrições de Use Cases para tratar as anomalias, definindo, assim, os possíveis métodos das pseudo-classes (candidatas a classe) do sistema, obtidas pelo padrão Criar Visão OO dos Dados.

5. Identificar Relacionamentos

Intuito:

- Identificar os relacionamentos, seus tipos e armazená-los como fatos na Base de Conhecimento.

Problema:

- Pela falta de documentação existente, não se sabe como os arquivos de dados do sistema legado estão relacionados entre si.

Influências:

- Sistemas implementados em linguagens como Clipper, Cobol, entre outras, em geral não utilizam banco de dados relacional, no qual os relacionamentos entre as tabelas são explícitos.
- A partir da inspeção dos dados e da análise do código fonte é possível identificar o relacionamento dos dados no sistema legado.

Solução:

Os supostos relacionamentos são reconhecidos através de variáveis e comandos, embutidos nos transformadores, que manipulam dados de um arquivo ou de uma tabela de um banco de dados. É verificado se um campo de um arquivo ou tabela, identificado como chave, tem seus valores consultados e atribuídos a uma variável, e posteriormente o valor dessa

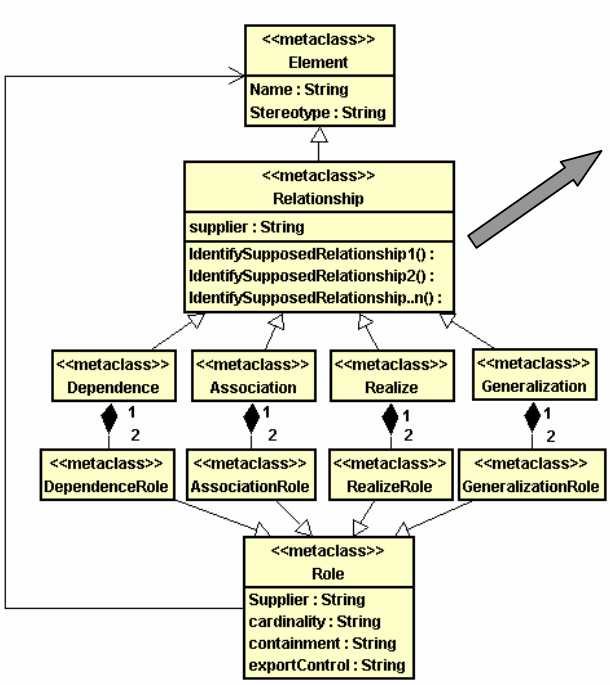
variável é armazenado em campos de outro arquivo ou tabela ou é utilizado para fazer busca em registros de outros arquivos ou tabelas. São armazenadas informações na Base de Conhecimento, como: o nome do arquivo, tabela de origem e de destino, o nome do relacionamento na origem e no destino.

É necessária a interação do Engenheiro de Software para informar os tipos dos relacionamentos.

Um relacionamento pode ser do tipo:

- ***"é parte de"***, se um arquivo ou tabela complementa outro arquivo ou tabela,
- ***"é um"***, se um arquivo ou tabela compartilha a estrutura de outro arquivo ou tabela, ou
- ***"está associado"***, se existe navegabilidade entre os arquivos ou tabelas relacionados.

Utiliza-se de transformação para reconhecer e armazenar na Base de Conhecimento esses supostos relacionamentos e através de interações com o Engenheiro de Software informar o tipo de relacionamento mais apropriado para o suposto relacionamento reconhecido durante as transformações, conforme mostra a Figura 7, usando o metamodelo.




```
TRANSFORM
IdentifySupposedRelationship1
LHS: {{dast clipper.statements
     SEEK [[expr_num field]]
}}
POST-MATCH: {{dast txt.decls
...
Get_Atrib(FileName);
...
fprintf(stderr," ** RELAC IDT **");
fprintf(stderr,"O tipo do
relacionamento deve ser informado");
fprintf(stderr,"\n %s e parte de %s
?", class, classcorrente);
cin >> decisao;
if ((strcmp(decisao, "S") == 0)) {
     sprintf(rel_tipo, "\"%s\"","é
     parte de");
}
...
KBAssertIfNew("Relationship([[elemen
t1]],[[element2]], [[relationship]]
,[[cardinality]]");
KBWrite("UtilityRss.kb");
...
}}
```

Figura 7. Transformação para identificação de supostos relacionamentos

Padrões Relacionados:

- Este padrão é a entrada para o padrão Criar Especificações dos Relacionamentos e Cardinalidades, utilizado no passo Recuperar.
- Este padrão está relacionado com o padrão ***"Identificar Relacionamentos"*** proposto por [14]: que são identificados através da tabela Entidades x Chaves e trechos do código fonte que auxiliaram na elaboração dessa tabela.

6. Identificar Cardinalidades

Intuito:

- Identificar as cardinalidades dos relacionamentos.

Problema:

- Com a pouca documentação existente dos sistemas legados, se faz necessário analisar as diferentes estruturas de dados a fim de identificar o grau das cardinalidades.

Influências:

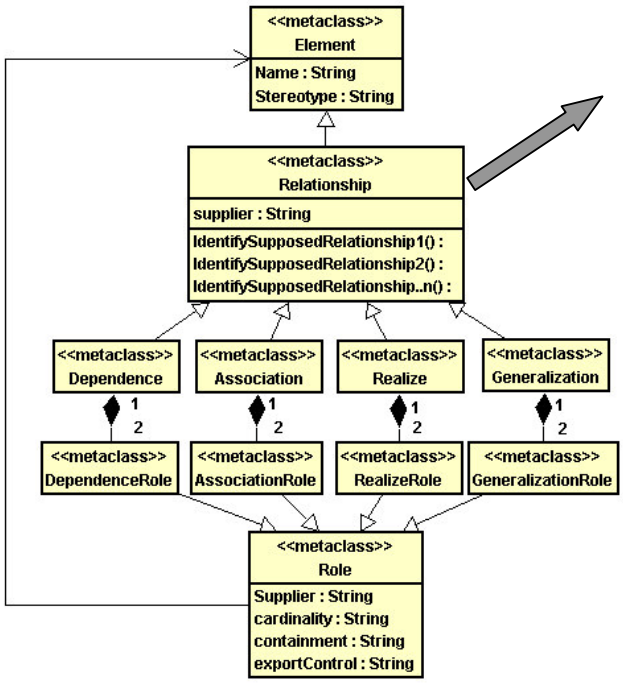
- Fazer uma análise em arquivos ou tabelas não normalizadas.
- Interação do Engenheiro de Software para validar as cardinalidades.

Solução:

Para determinar a cardinalidade de um relacionamento, é necessário analisar os campos dos arquivos ou tabelas que dão origem ao relacionamento e consultar as informações dos campos desses arquivos ou tabelas, registro a registro, para identificar o número de ocorrências de um mesmo valor armazenado.

Se um dado valor no campo de um arquivo ou tabela, ocorrer repetidas vezes significa que a cardinalidade máxima do arquivo ou tabela é ***"n"***.

No entanto, as informações obtidas dos arquivos ou tabelas podem não ser suficientes para definir as verdadeiras cardinalidades de um relacionamento, necessitando da interação do Engenheiro de Software. Essa interação poderá ser feita durante as transformações. A Figura 8 mostra uma transformação que permite ao Engenheiro de Software inserir as cardinalidades dos relacionamentos. O Engenheiro de Software interage para confirmar os valores dessas cardinalidades e atualizar os fatos de relacionamentos (*Relationship*).




```
TRANSFORM IdentifyCardinality1
LHS: {{dast clipper.statements
      SEEK [[expr_num field]]
}}
POST-MATCH: {{dast txt.decls
...
Get_Atrib(FileName);
...
fprintf(stderr,"Arquivos de dados:
existe o relacionamento: ");
fprintf(stderr,"%s está relacionado
com %s %s", classe, destino, classcc);
fprintf(stderr,"%s está relacionado
com %s %s", classcc, origem,classe);
fprintf(stderr,"está correto? ");
cin >> decisao;
if ((strcmp(decisao, "S") == 0)) {
sprintf(query, "Relationship(*a, *b,
*c, *d)", cardinality);}
...
KBAssert("Relationship([[element1]],
        [[element2]],[[relationship]],
          [[cardinality]])");
     KBWrite("UtilityRss.kb");
  ...
  }}
```

Figura 8. Transformação para identificação de cardinalidades

Padrões Relacionados:

- Este padrão é a entrada para o padrão Criar Especificações dos Relacionamentos e Cardinalidades, utilizado no passo Recuperar.

Passo Organizar

O objetivo deste passo é organizar o código legado, em classes, atributos e métodos.

Com base nas informações identificadas no passo anterior (Identificar) armazenadas na Base de Conhecimento, utilizando de transformações apropriadas que deverá segmentar o código legado, organizando-o em classes, com seus atributos e métodos, produzindo um código intermediário, ainda na mesma linguagem do código legado.

Para atingir este objetivo o Engenheiro de Software deverá seguir os padrões que estão detalhados a seguir:

1. Criar Supostas Classes e Supostos Atributos

Intuito:

- Criar supostas classes e supostos atributos.

Problema:

- É difícil reconhecer supostas classes e supostos atributos em um sistema legado procedimental. Uma classe pode ser constituída de *n* programas do código legado.

Influências:

- Os fatos armazenados na Base de Conhecimento auxiliam o Engenheiro de Software na construção das supostas classes e atributos.

Solução:

As supostas classes identificadas no passo anterior darão origem a classes, através das transformações de organização. Para cada classe deve-se criar um arquivo com o mesmo nome da classe, onde serão alocados todos os seus atributos e métodos.

Os supostos atributos identificados no passo anterior devem originar atributos. Portanto, tem-se a interação do Engenheiro de Software para informar se cada um destes supostos atributos será realmente um atributo da classe.

Esta tarefa é realizada visando garantir a integridade nas informações pertinentes aos atributos, e diminuir a possibilidade de criação automática de atributos que não são necessários. Como vimos no passo Identificar, os fatos de supostos atributos são oriundos dos campos das tabelas, de arquivos de dados ou de bancos de dados do código legado. Existem recursos de implementação como os contadores, dentre outros, que não precisam ser recuperados como atributos no projeto do código legado, então o Engenheiro de Software poderá, durante essa interação, impedir a criação destes atributos, caso seja necessário.

A Figura 9 mostra o programa “Clientes.prg” organizado de acordo com os fatos identificados e armazenados na Base de Conhecimento, no passo anterior, Identificar.

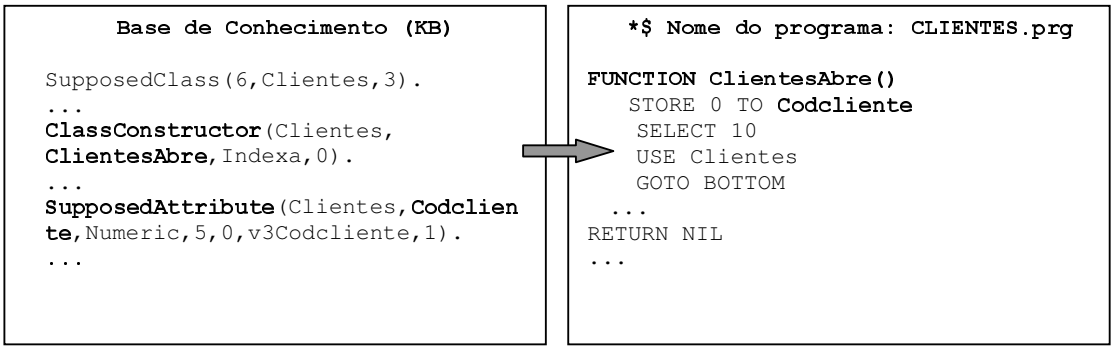


Figura 9. Organização da Classe Clientes de acordo com os fatos armazenados na KB

Padrões Relacionados:

- Este padrão tem como entrada o padrão Identificar Supostas Classes e Atributos, do passo Identificar, e é utilizado como entrada para o padrão Criar Especificações das Classes e dos Atributos no passo Recuperar.

2. Alocar Supostos Métodos nas Supostas Classes

Intuito:

- Criar os supostos métodos nas supostas classes criadas.

Problema:

- A desorganização do código legado com relação às funções, *procedures* e métodos, que futuramente pertencerão a apenas uma classe, dificulta a alocação dos supostos métodos.

Influências:

- A forma ou maneira como os supostos métodos acessam os dados influenciam na sua alocação.
- A identificação feita anteriormente pelo transformador e o armazenamento dos fatos na Base de Conhecimento, facilita a correta alocação de métodos nas supostas classes, através das transformações.

Solução:

O objetivo desta tarefa é alocar nos arquivos das classes, os seus respectivos supostos métodos identificados no passo, Identificar, corrigindo-se anomalias da seguinte forma:

Os supostos métodos são classificados em construtores (c), quando alteram a estrutura de dados, e observadores (o), quando somente consultam as estruturas de dados. Assim, quando um suposto método refere-se a mais de uma suposta classe ele deve ser alocado usando os seguintes critérios [09]:

- Se construtor de uma suposta classe e observador de outra (oc), será alocado na suposta classe que faz referência como construtor;
- Se observador de uma suposta classe e construtor de várias outras classes (oc+), será alocado na primeira suposta classe que faz referência como construtor;
- Se observador de mais de uma suposta classe e construtor de apenas uma (o+c), será alocado na primeira suposta classe que faz referência como observador.

O Engenheiro de Software poderá utilizar um Sistema de Transformação para realizar esta tarefa. As transformações deverão consultar a Base de Conhecimento (KB) para obter informações sobre os supostos métodos e alocá-los na workspace da suposta classe a qual esse suposto método pertence.

A Base de Conhecimento é consultada para obter informações sobre os supostos métodos, e as anomalias não solucionadas pelas transformações são resolvidas pelo Engenheiro de Software, que pode tomar decisões, interagindo com o Sistema de Transformação durante a aplicação das transformações.

Padrões Relacionados:

- Este padrão tem como entrada o padrão Identificar Supostos Métodos, do passo Identificar, e é utilizado como entrada para o padrão Criar Especificações dos Métodos no passo Recuperar.

Passo Recuperar

O objetivo deste passo é recuperar as especificações numa linguagem de Modelagem, obtendo assim o Projeto Orientado a Objetos Recuperado do Código Legado.

O Engenheiro de Software, com o auxílio de transformações e de um Sistema Transformacional, parte do Código Legado Organizado, obtido no passo Organizar, para gerar essas especificações do projeto, descritas na linguagem de modelagem alvo. Depois de geradas as especificações, estas podem ser importadas pelo Engenheiro de Software numa ferramenta CASE para poder visualizar o projeto recuperado graficamente.

Os padrões utilizados para a geração das especificações do projeto são apresentados a seguir.

1. Criar Especificações das Classes e dos Atributos

Intuito:

- Gerar especificações na linguagem de modelagem para as classes e atributos obtidos no passo anterior.

Problema:

- As informações sobre as classes e atributos foram identificadas e organizadas, mas ainda precisam ser convertidas para uma linguagem de modelagem, na qual requer regras sintáticas.

Influências:

- O trabalho de conversão do código legado para o código OO, normalmente, é repetitivo, e está sujeito a erros. Esse padrão visa a automatização do processo de conversão das classes e seus respectivos atributos.

Solução:

Através de transformações, criar as especificações na linguagem de modelagem MDL (*Modelling Domain Language*), para as classes e seus atributos. Outras linguagens de modelagem poderiam ser utilizadas, como: Notação do Coad/Yourdon [25], OMT [26], Booch [24], etc. Para realizar esta atividade deve-se:

- Para cada classe, criar a especificação correspondente na linguagem de modelagem*;*
- Para cada atributo da classe, criar a especificação correspondente na linguagem de modelagem;

Cada unidade de programa do sistema legado, organizada como uma classe, dá origem a sua especificação na linguagem de modelagem. O mesmo acontece para cada atributo da unidade de programa reconhecido, dando origem a sua respectiva especificação na linguagem de modelagem. A Figura 10 mostra uma parte do transformador que reconhece as unidades de programa (funções, procedimentos, programas) do sistema legado, para gerar as especificações de classes e de seus atributos na linguagem MDL.

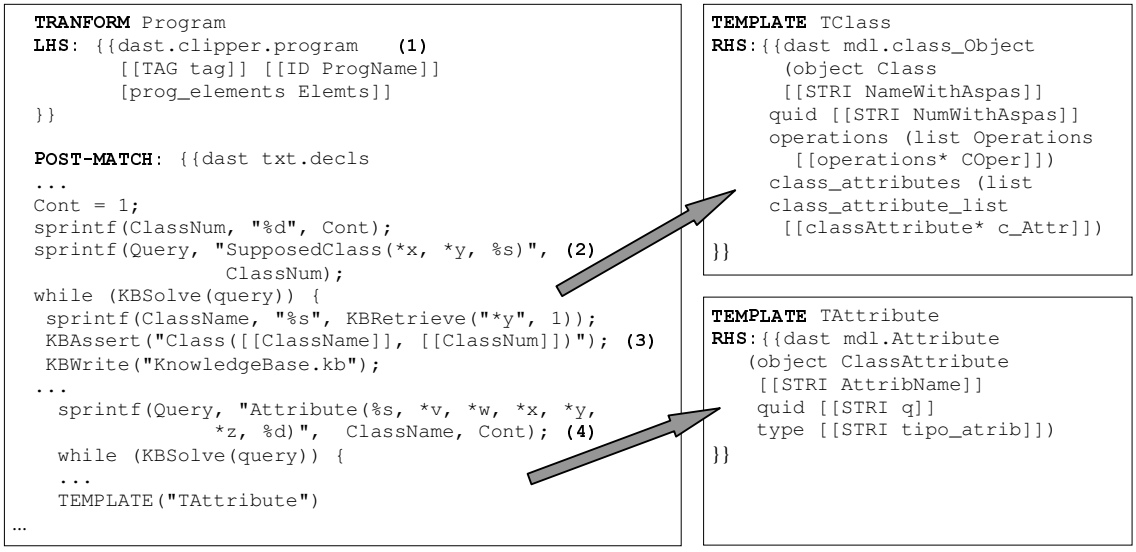


Figura 10. Transformação que gera a especificação na linguagem de modelagem para a classe e seus atributos

Padrões Relacionados:

- Este padrão tem como entrada o padrão Criar Supostas Classes e Supostos Atributos, do passo Organizar.
- Relaciona-se com os padrões "***Definir as Classes***" e "***Definir os Atributos***" proposto por [14], que apenas consolidam diversos diagramas já obtidos numa apresentação segundo a visão de Orientação a Objetos.

2. Criar Especificações dos Métodos

Intuito:

- Gerar as especificações dos métodos na linguagem de modelagem.

Problema:

- Com os supostos métodos já identificados e armazenados na base de conhecimento, é preciso pesquisar na linguagem de modelagem a estrutura de um método, para que se possa gerar suas especificações neste formato, seguindo suas regras sintáticas.
- A inclusão do corpo do método já na linguagem alvo da reimplementação, na especificação do método, completa o futuro projeto recuperado do código legado.

Influências:

- O trabalho de conversão do código legado para o código OO, normalmente, é repetitivo e está sujeito a erros. Esse padrão visa a automatização do processo de conversão dos métodos.

Solução:

Construir transformadores para criar as especificações das assinaturas dos métodos e transformar os corpos dos métodos diretamente para uma linguagem de programação alvo da reimplementação.

Assim como é feito para as classes e seus respectivos atributos, a especificação dos métodos também é obtida com o auxílio de um Sistema Transformacional. A Figura 11 mostra, como exemplo, a criação das especificações de um método na linguagem de modelagem.

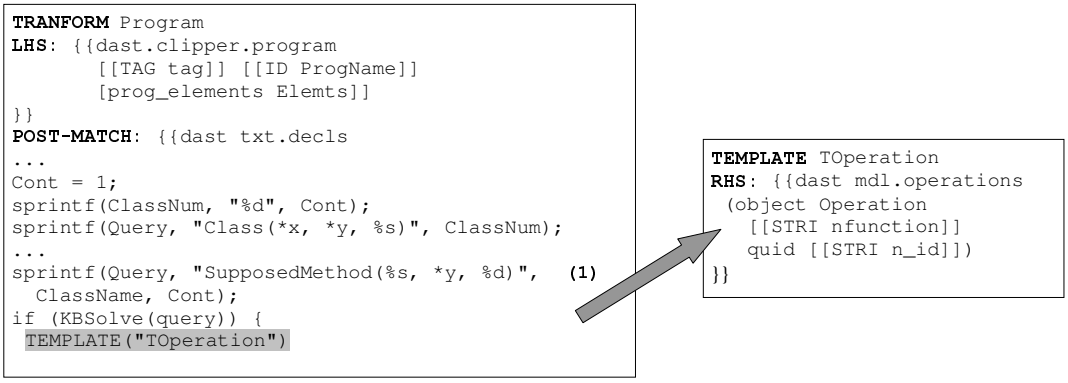


Figura 11. Transformação que gera a especificação na linguagem de modelagem para um método

No exemplo da Figura 11, para cada fato *"SupposedMethod"* é criada a sua correspondente especificação na linguagem de modelagem (1) através da *TEMPLATE "TOperation"*.

Padrões Relacionados:

- Este padrão tem como entrada o padrão Alocar Supostos Métodos nas Supostas Classes, do passo Organizar.
- Relaciona-se com o padrão ***"Definir Métodos"*** proposto por [14], que obtém os métodos das classes no Diagrama de Classes em construção por meio da descrição do Use Case correspondente.

3. Criar Casos de Uso

Intuito:

- Utilizar transformações para criar os casos de uso.
- Identificar os fatos sobre os supostos casos de uso armazenados na base de conhecimento para a geração das especificações correspondentes na linguagem de modelagem.

Problema:

- Os casos de uso já foram identificados e armazenados na base de conhecimento como fatos, mas ainda precisam ser convertidas na linguagem de modelagem, respeitando suas regras sintáticas.

Influências:

- O trabalho de conversão do código legado para o código OO, normalmente, é repetitivo e está sujeito a erros. Esse padrão visa a automatização do processo de conversão dos fatos aramazenados em Base de Conhecimento em Modelos de Caso de Uso.

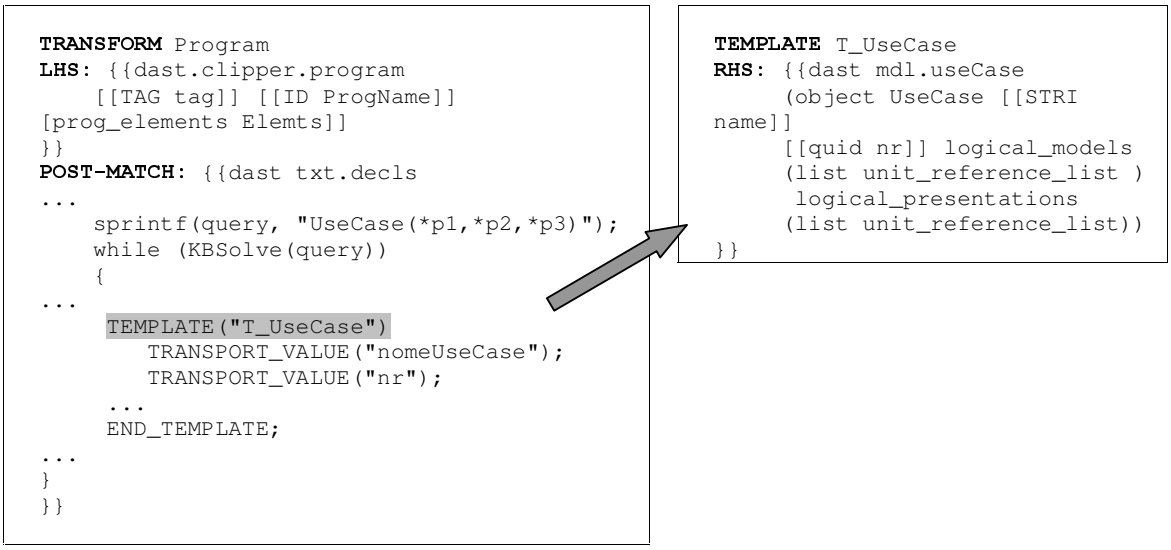
Solução:

Criar as especificações dos Casos de Uso a partir dos fatos recuperados da Base de Conhecimento.

Para cada suposto Caso de Uso já identificado, deve-se gerar especificações correspondentes na Linguagem de Modelagem, observando as seguintes recomendações:

- Para cada Caso de Uso identificado, criar as respectivas especificações na linguagem de modelagem;
- Para cada ator identificado, criar especificações de classe com "stereotype" pré-definido para a identificação de atores na linguagem de modelagem;

A Figura 12 mostra, como exemplo, a criação das especificações de um Use Case através da *TEMPLATE T_UseCase*, a partir de um fato *UseCase* da Base de Conhecimento.



```
TRANSFORM Program
LHS: {{dast.clipper.program
    [[TAG tag]] [[ID ProgName]]
[prog_elements Elemts]]
}}
POST-MATCH: {{dast txt.decls
...
    sprintf(query, "UseCase(*p1,*p2,*p3)");
    while (KBSolve(query))
    {
...
     TEMPLATE("T_UseCase")
        TRANSPORT_VALUE("nomeUseCase");
        TRANSPORT_VALUE("nr");
     ...
     END_TEMPLATE;
...
}
}}
```

```
TEMPLATE T_UseCase
RHS: {{dast mdl.useCase
     (object UseCase [[STRI
name]]
     [[quid nr]] logical_models
     (list unit_reference_list )
      logical_presentations
     (list unit_reference_list))
}}
```

Figura 12. Transformação de recuperação das especificações de supostos Casos de Uso

Padrões Relacionados:

- Este padrão tem como entrada os padrões Identificar Casos de Uso e Definir hierarquia chama/chamado, do passo Identificar.
- Relaciona-se com o padrão ***"Construir Diagramas de Use Cases"*** proposto por [14], que constroem os diagramas de casos de uso do sistema a partir da tabela de cenários do sistema elaborada pelo padrão Obter Cenários [14]. Este padrão é executado em conjunto com o padrão "***Interview During Demo"*** proposto por [13], entrevistar o usuário durante o sistema em Operação.

4. Criar Diagrama de Seqüência

Intuito:

- Identificar o fluxo de controle do código legado organizado e dos casos de uso para a geração do diagrama de seqüência.

Problema:

- O fluxo identificado no passo, Identificar, representa o fluxo entre as unidades de programas, e não um fluxo entre os objetos, portanto representa um diagrama de seqüência procedural.

Influência:

- O trabalho de conversão do código legado para o código OO, normalmente, é repetitivo e está sujeito a erros. Esse padrão visa a automatização do processo de conversão dos fatos aramazenados em Base de Conhecimento em Diagramas de Sequencia.

Solução:

Baseado nos Casos de Uso, para cada seqüência de chamada de cada caso de uso cria-se um diagrama de seqüência.

Cada diagrama de seqüência corresponde a um curso, Normal ou Alternativo, do caso de uso. O fluxo de controle do Código Legado Organizado, definido pela seqüência de chamadas das unidades de programas, é mantido através das conexões de mensagens entre os objetos.

Padrões Relacionados:

- Este padrão utiliza-se dos padrões Definir hierarquia chama/chamado e Identificar Casos de Uso, do passo Identificar, para realizar sua tarefa.
- Relaciona-se com o padrão ***"Construir Diagramas de Seqüência"*** proposto por [14], que objetiva a construção dos Diagramas de Seqüência, a partir de cada descrição de Use Case, obtida pelo padrão Elaborar Descrição de Use Cases.

5. Criar Especificações dos Relacionamentos e Cardinalidades

Intuito:

- Gerar as especificações dos relacionamentos e suas cardinalidades na linguagem de modelagem, baseando-se nos fatos armazenados na base de conhecimento.

Problema:

- Conhecer as regras sintáticas das especificações que representam os relacionamentos e suas cardinalidades na linguagem de modelagem.

Influência:

- Os fatos armazenados na Base de Conhecimento, identificados pelos padrões Identificar Cardinalidades e Identificar Relacionamentos, auxiliam na correta especificação dos relacionamentos e cardinalidades.

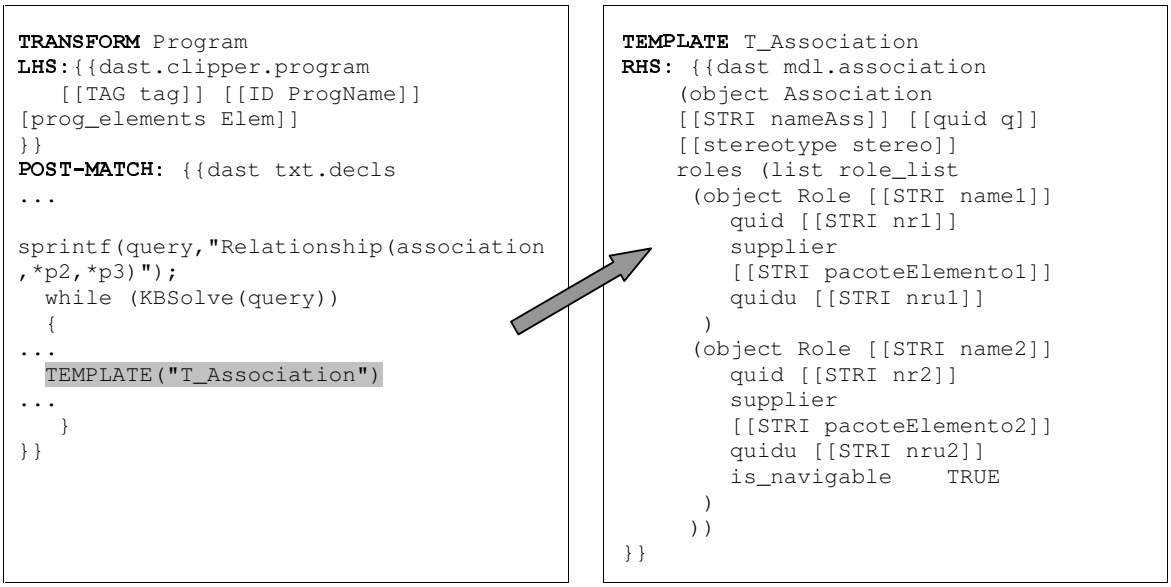
Solução:

Criar as especificações dos relacionamentos com suas respectivas cardinalidades, baseado nos fatos armazenados na Base de Conhecimento.

- Para cada suposto relacionamento já identificado, deve-se gerar especificações correspondentes na linguagem de modelagem, acoplando-se suas respectivas cardinalidades também já identificadas;

Assim, são criadas, para cada relacionamento, as especificações na linguagem de modelagem contendo informações da classe de origem e classe de destino, com suas cardinalidades.

A Figura 13 mostra, por exemplo, uma transformação que reconhece na Base de Conhecimento um fato *Relationship* de uma associação e gera sua respectiva especificação na linguagem de modelagem MDL através da *TEMPLATE T_Association.*



```
TRANSFORM Program
LHS:{{dast.clipper.program
   [[TAG tag]] [[ID ProgName]]
[prog_elements Elem]]
}}
POST-MATCH: {{dast txt.decls
...

sprintf(query,"Relationship(association
,*p2,*p3)");
  while (KBSolve(query))
  {
...
  TEMPLATE("T_Association")
...
   }
}}
```

```
TEMPLATE T_Association
RHS: {{dast mdl.association
    (object Association
    [[STRI nameAss]] [[quid q]]
    [[stereotype stereo]]
    roles (list role_list
     (object Role [[STRI name1]]
        quid [[STRI nr1]]
        supplier
        [[STRI pacoteElemento1]]
        quidu [[STRI nru1]]
      )
     (object Role [[STRI name2]]
        quid [[STRI nr2]]
        supplier
        [[STRI pacoteElemento2]]
        quidu [[STRI nru2]]
        is_navigable    TRUE
      )
     ))
}}
```

Figura 13. Transformação de recuperação das especificações de um relacionamento

Padrões Relacionados:

- Este padrão tem como entrada o padrão Identificar Relacionamentos e Identificar Cardinalidades, do passo Identificar.

Usos Conhecidos dos Padrões Propostos:

Finalmente, o item **Usos Conhecidos** é explicitado a seguir, por ser comum a todos os padrões.

Os conceitos aqui apresentados foram utilizados nos processos de engenharia reversa realizados em [15, 16, 17, 18, 19, 20, 21, 22], sem pertencerem a um grupo de Padrões.

Em tais estudos e experiências foram utilizados dois mecanismos para auxiliá-los no processo de Engenharia Reversa: o ST Draco-PUC[11] e a ferramenta MVCase[02]. A Figura 14 exemplifica todo o processo com as técnicas e mecanismos utilizados no processo de Engenharia Reversa.

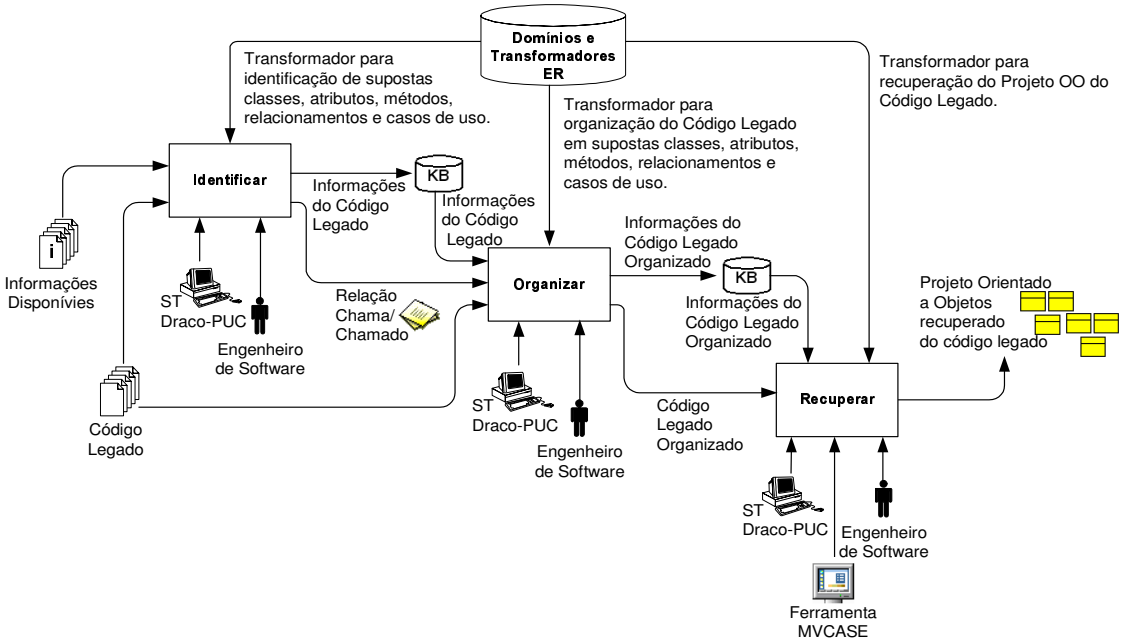


Figura 14. Padrões de Processo para a Engenharia Reversa.

Agrupando os Padrões

Após a visão geral dos padrões descritos nas seções anteriores, a Figura 15 mostra a interação destes padrões, provendo um conjunto de Padrões de Processo para a Engenharia Reversa baseada em Transformações.

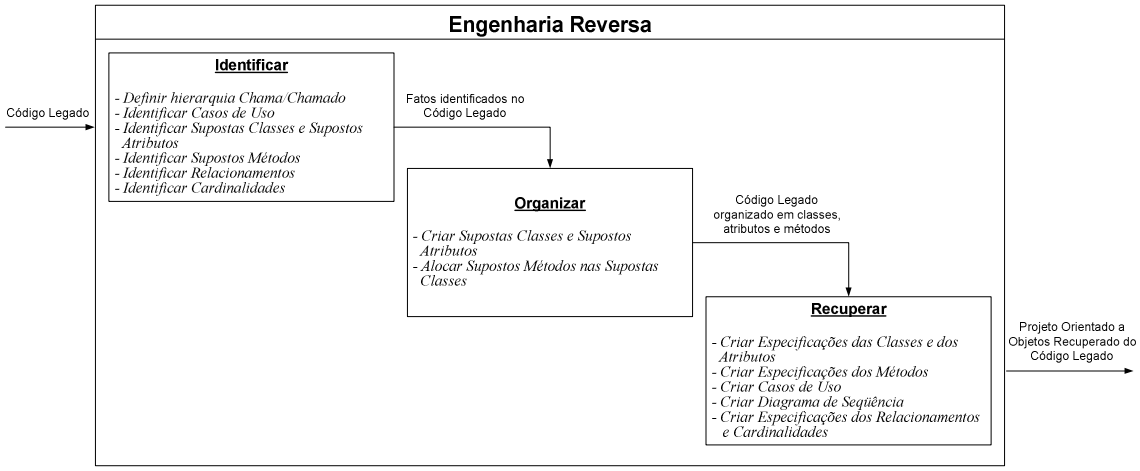


Figura 15. Interação dos Padrões de Processo para a Reengenharia Reversa.

Na Engenharia Reversa, o Engenheiro de Software parte do *Código Legado* e suas informações disponíveis, para obter o Projeto Orientado a Objetos Recuperado do Código Legado e as Especificações MDL do Projeto Orientado a Objetos. O projeto Orientado a Objetos é representado por modelos gráficos e textuais, por exemplo usando a notação UML.

As especificações do Projeto Orientado a Objetos são descritas na linguagem de modelagem MDL. Estas descrições em MDL permitem recuperar o projeto Orientado a Objetos e dar continuidade ao processo de reengenharia.

Agradecimentos

Os autores gostariam de agradecer a Prof.ª Drª. Rosana Teresinha Vaccare Braga pelas suas orientações durante o processo de s*hepherding*.

Referências

[01] Leite, J.; Sant'Anna, M.; Freitas, F.; **Draco-PUC: A Technology Assembly For Domain Oriented Software Development.** Proceedings of ICSR94 (International Conference on Software Reuse), IEEE Press, 1994.

[02] Almeida, E., S., Lucrédio, D., Bianchini, C., P., Prado, A., F., Trevelin, L., C., 2002. **MVCase Tool: An Integrating Technologies Tool for Distributed Component Development (in portuguese).** In SBES'2002, 16th Brazilian Symposium on Software Engineering, Tools Session.

[03] D'Souza, D., F., Wills, A., C., 1999. **Objects, Components, and Frameworks with UML, The Catalysis Approach,** Addison-Wesley. USA.

[04] Gamma, E., et al., 1995. **Elements of Design Patterns: Elements of Reusable Object Oriented Software**, Addison-Wesley.

[05] Werner, C.M.L.; Braga, R. M.M. **Desenvolvimento Baseado em Componentes.** XIV Simpósio Brasileiro de Engenharia de Software SBES2000 Minicursos e Tutoriais – pg. 297-329 – 2-6 de Outubro, 2000.

[06] Booch, G. et al. **The Unified Modeling Language.** User Guide. USA: Addison Wesley, 1999.

[07] Fowler R, M. **UML Destilled. Applying the Standard Object Modeling Language.** England:Addison Wesley, 1997.

[08] **Unified Modeling Language 1.4 specification**. Available at site Object Management Group. URL: http://www.omg.org/technology/documents/formal/uml.htm. Consulted in February, 2003.

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

[10] Neighbors, J.M., **Software Construction Using Components**, Doctoral dissertation, Information and Computer Science Dept. University of California, Irvine, 1980.

[11] 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.

[12] Almeida, E., S., Bianchini, C., P., Prado, A., F., Trevelin, L., C., 2002. **MVCase: An Integrating Technologies Tool for Distributed Component-Based Software Development.** In APNOMS'2002, The Asia-Pacific Network Operations and Management Symposium,Poster Session. Proceedings of IEEE.

[13] Demeyer, S.; Ducasse, S.; Nierstrasz, O., **"A Pattern Language for Reverse Engineering".** Proceedings of the 5th European Conference on Pattern Languages of Programming and Computing, (EuroPLOP'2000), Andreas Ruping(Ed.), 2000.

[14] Recchia, E. L.; Penteado, R. – **FaPRE/OO: Uma Família de Padrões para Reengenharia Orientada a Objetos de Sistemas Legados Procedimentais.** Artigo apresentado no SugarloafPLoP2002 – The Second Latin American Conference on Pattern Languages of Programming – Agosto/2002, Itaipava – RJ.

[15] Novais, E. R. A.; **Reengenharias de Software Orientadas a Componentes Distribuídos.** São Carlos/SP, 2002, Dissertação de Mestrado, Universidade Federal de São Carlos.

[16] Bossonaro, A. A.; **Estratégia de Reengenharia de Software usando Transformações**. Disponível em URL: http://www.recope.dc.ufscar.br.

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

Carlos.

[18] Fukuda, A. P. **Refinamento Automático de Sistemas Orientados a Objetos Distribuídos.** São Carlos/SP, 2000. Dissertação de Mestrado. Universidade Federal de São Carlos.

[19] Sametinger, J. **Software Engineering with Reusable Components**. Springer-Verlag, 1997.

[20] Garcia, V. C., Fontanette, V., Perez, A. B., Prado, A. F., Sant'Anna, M.; **RHAE/CNPQ - Projeto: Reengenharia de Software Usando Transformações (RST)**. Processo número: 610.069/01-2.

[21] Nogueira, A. R.; **Transformação de DataFlex Procedural para Visual DataFlex Orientado a Objetos reusando um Framework.** São Carlos/SP, 2002, Dissertação de Mestrado. Universidade Federal de São Carlos.

[22] Prado, A.F. **Estratégia de Engenharia de Software Orientada a Domínios**. Rio de Janeiro-RJ, 1992. Tese de Doutorado. Pontifícia Universidade Católica. 333p.

[23] Abrahão, S.M., Prado, A.F; Sant'anna, M.; **A Semi-Automatic Approach for Building Web-Enabled Aplications from Legacy.** Submitted on 4 IEEE international software engineering Standards Symposium – Curitiba, Brasil, May 1999.

[24] Booch, G., Rumbaugh, J., Jacobson, I; **The Unified Modeling Language Reference Manual**, Addison-Wesley, 1999.

[25] Coad, P., Yourdon, E. **Object-Oriented Design**. Yourdon Press/Prentice-Hall, Englewood Cliffs, NJ, 1991.

[26] Rumbaugh, J., Blaha, M., Premerlani, W. F. Eddy, and W. Lorensen. **Object-oriented modeling and design**. Prentice-Hall, 1991.