

Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Editor de Esquemas para SIRIUS
utilizando um Gerenciador Relacional
como Repositório de Dados
Documentação de Implementação

Myrian Renata Barros Araujo
Caetano Traina Jr.

Nº 91

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Abr./1999

SYSNO	1043597
DATA	/ /
ICMC - SBAB	

Instituto de Ciências Matemáticas e de Computação

Editor de Esquemas para SIRIUS
utilizando um Gerenciador Relacional
como Repositório de Dados

Myrian Renata Barros Araujo
e_mail: mrenata@icmc.sc.usp.br

Caetano Traina Jr.
e_mail: caetano@icmc.sc.usp.br

ICMC - USP
13560 - 970

Caixa Postal 668
São Carlos SP

Abril de 1999

Sumário

Capítulo 1	05
Introdução	05
1.1 Considerações iniciais	05
1.2 Organização do Trabalho	06
 Capítulo 2	 07
Base de Dados	07
 Capítulo 3	 12
Implementação	12
3.1 Rotinas de E ² SIRIUS	14
3.2 Interadores	27
3.3 Ação(ões) associada(s) ao evento	32
 Capítulo 4	 38
Considerações finais	38
 Anexo	 40
Código do Formulário Objeto	40
 Referências Bibliográficas	 65

Lista de Quadros e Figuras

Quadro 1 - Inicialização de dados do esquema	08
Quadro 2 - Procedure para geração de OIDs	09
Quadro 3 - Relação referente a Abstração de Classificação	09
 Figura 1 - Relação Objeto	 10
Figura 2 - Relações da Abstração de Agregação	10
Figura 3 - Esquema Relacional para SIRIUS	11
Figura 4 - Formulário Objeto	12

Capítulo 1

Introdução

1.1 Considerações iniciais

Este relatório contém aspectos da implementação do Editor de Esquemas - E²SIRIUS [Araujo_1999]. E²SIRIUS foi projetado para dar suporte a Metamodelos Orientados a Objetos. O aplicativo foi desenvolvido como parte do trabalho de mestrado [Araujo_1998], [Araujo_1998a], contando com a colaboração da então aluna de iniciação científica Elaine Machado Parros. Os conceitos utilizados na construção deste editor aplicam-se a outros modelos elaborados segundo o paradigma de orientação a objetos que tenham a característica de serem metamodelos. Em particular, o editor desenvolvido apoia-se no modelo SIRIUS [Biajiz_1996].

O modelo SIRIUS é o módulo centralizador de um sistema mais abrangente, o Sistema SIRIUS, que está sendo desenvolvido pelo grupo GbdI - Grupo de Base de Dados e Imagens do ICMC-USP [<http://www.gbdi.icmc.sc.usp.br/documentacao/apostilas/sirius/>]. Este sistema é composto basicamente por três módulos:

- o Modelo SIRIUS;
- o Gerenciador SIRIUS/GO [Traina_1996];
- e as Técnicas SIRIUS de Desenvolvimento de Sistemas.

A confecção do editor teve como objetivo oferecer suporte tanto para a edição de modelagens de situações reais no Gerenciador de Objetos SIRIUS/GO, quanto auxiliar projetistas no aprendizado e treinamento dos conceitos do modelo. Como Editor de Esquemas de SIRIUS/GO, fornece suporte necessário para a especificação detalhada de todos os conceitos do modelo, bem como, recursos para a alimentação de dados de teste em pequenos volumes, tendo também uma interface que procura facilitar o trabalho de concepção e digitação de esquemas. Como ferramenta de auxílio à aprendizagem do modelo, o Editor apresenta recursos para permitir a manipulação e visualização de conceitos próprios de SIRIUS.

1.2 Organização do Trabalho

Estes relatório está organizado em 4 capítulos:

- o Capítulo1: descreve de modo geral o Editor de Esquemas, contextualizando-o no Sistema SIRIUS;
- o Capítulo2: descreve como foi elaborada a base de dados que dá suporte para a criação do editor;
- o Capítulo 3: enfoca aspectos de implementação;
- o Capítulo 4: contém as considerações finais.

Capítulo 2

Base de Dados

E²SIRIUS foi elaborado para atender as necessidades de um Metamodelo Orientado a Objetos, portanto este foi desenvolvido de modo a disponibilizar, de forma homogênea, os conceitos de objetos tipo e objetos que não são tipos. A concepção do editor de esquemas desta maneira é fundamental, pois este é baseado em um metamodelo, sendo assim é relevante obter informações não só da base de dados (base extencional) mais também do esquema da modelagem (base intencional).

O processo empregado para a elaboração de E²SIRIUS com a finalidade de obter a homogeneidade de acesso a objetos tipo e não tipo foi o mapeamento do modelo SIRIUS para o modelo relacional. Optou-se por utilizar um núcleo relacional por questões práticas, uma vez que a versão nativa do Gerenciador de Objetos SIRIUS/GO, baseada em um núcleo desenvolvido para dar suporte a gerenciadores de objetos, encontra-se em fase de desenvolvimento por membros do Grupo de Banco de Dados e Imagens do ICMC-USP.

Entretando, devido ao conjunto de representação formal existente em SIRIUS, o mapeamento para o modelo relacional foi feito de modo coerente, uma vez que todos os construtores semânticos do modelo já estavam previstos no formalismo, possibilitando, deste modo, que o mapeamento para o modelo relacional fosse feito de maneira consistente. Levou-se também em consideração algumas regras de controle de consistência durante a elaboração do Editor, fundamentais para a criação de uma base de dados que irá armazenar informações das modelagens feitas utilizando-se o modelo. Isto foi possível devido à precisão e ao rigor de expressões algébricas utilizadas na representação formal do modelo SIRIUS.

O repositório de dados para E²SIRIUS foi construído, com o auxílio do gerenciador relacional INTERBASE versão 4.2, de maneira portátil, utilizando-se os recursos básicos da linguagem SQL, de maneira que o aplicativo possa ser conectado em outro servidor, monousuário ou multiusuário, que utilize o padrão SQL. A base de dados para E²SIRIUS foi criada a partir de um conjunto de três *scripts*: `tabelas.sql`, `inicializações.sql` e `código.sql`.

Como SIRIUS é um metamodelo, houve a necessidade de uma pré-inicialização para que as

informações a respeito do metamodelo fossem armazenadas na base. Isto foi feito através do *script* inicializações. É a partir destas que os dados do usuário são armazenados na base. Desta forma, tais dados pré-inicializados sempre estarão contidos na base, mesmo quando esta não tiver dados de modelagens de usuários. O Quadro 1, parte do arquivo *inicializações.sql* exibe algumas inicializações para dados do metamodelo.

```
INSERT INTO objeto (CodObj, CodTObj) VALUES (1, 1);
/* Meta Tipo Objeto */
INSERT INTO objeto (CodObj, CodTObj) VALUES (2, 1);

/* Meta tipo atributo */
INSERT INTO tipo_atributo (CodTAt, NomeAt) VALUES (3, "Meta Tipo Atributo");

/* Vincula Meta tipo atributo a meta objeto */
INSERT INTO vincula_atributo_objeto (CodTAt, CodObj, Nivel, Identificador )
VALUES (3, 2, 0, "S");

/* Nome do Tipo Colonia Esquema*/
INSERT INTO tipo_colonia (CodTCol, NomeCol, CodObjConstr)
VALUES (10, "Esquema", 1);

/* Colonia do Tipo Esquema*/
INSERT INTO instancia_colonia (CodCol, CodTCol, CodObjConstr)
VALUES (11, 10, 1);

/* Nome do Tipo Colonia Global*/
INSERT INTO tipo_colonia (CodTCol, NomeCol, CodObjConstr)
VALUES (12, "Global", 1);

/* Colonia do Tipo Global*/
INSERT INTO instancia_colonia (CodCol, CodTCol, CodObjConstr)
VALUES (13, 12, 1);
```

Quadro 1 - Inicializações de dados do esquema.

A base de dados armazenar também os OID's (identificadores de objetos, tipos de objetos, atributos, tipos de colônia etc.). Estes são representados na implementação pelos códigos de objeto, tipo de objeto, atributos, tipo de colônia, entre outros. Os códigos são gerados automaticamente pelo sistema toda vez que o usuário criar qualquer objeto ou qualquer tipo de objeto. Estes são gerados no *script* código por meio de uma *stored procedure*. Esta função, apresentada no Quadro 2, gera os códigos automaticamente. Ela possui dois parâmetros de entrada: o primeiro é uma variável e o segundo é um incremento inteiro. Esta função, sempre que chamada, incrementa a variável com o incremento de entrada da função, retornando o valor da variável somada ao incremento.

```
SET TERM !!;
CREATE PROCEDURE gerador_codigo
RETURNS (codigo_retorno INT)
AS
BEGIN
    codigo_retorno = GEN_ID(contador, 1);
    EXIT;
END !!
SET TERM ;!!
COMMIT;
```

Quadro 2 - Procedure para geração de OIDs..

O *script* tabelas.sql contém a criação de todas as relações criadas a partir do mapeamento do modelo SIRIUS para o modelo relacional. O Quadro 3, por exemplo, exibe a relação criada no modelo relacional que mapeia os dados para a Abstração de Classificação do modelo SIRIUS.

```
CREATE TABLE objeto
(
    CodObj          INTEGER NOT NULL,
    CodTObj          INTEGER NOT NULL,
    CodColHab        INTEGER,
    CodTCollInst     INTEGER,
    CodCrit          INTEGER,

    PRIMARY KEY (CodObj)
);
```

Quadro 3 - Relação referente a Abstração de Classificação

O mapeamento para o modelo relacional foi feito por meio de criação de relações que cobrissem todo o embasamento teórico do modelo. Foram criadas várias relações a partir de uma principal, que é a relação *objeto* (Quadro 3). Esta agrega parte de todas as abstrações que formam o modelo. Como ilustrado na Figura 1, tem-se a relação *objeto*, formada por seis itens onde: o primeiro item (CodObj - código do objeto) está incluso naturalmente em todas as abstrações. Os três primeiros itens (CodObj, CodTObj - código tipo objeto, CodTAtIdent - código tipo do atributo identificador) formam uma base para representar a abstração de classificação. O quarto item (CodCrit - código critério) juntamente com o item CodObj formam uma base referencial para a abstração de generalização. Os dois últimos itens (CodColHab - código colônia habita, CodTCollInst - código do

tipo da colônia onde habitam as intâncias do objeto corrente) juntamente com o CodObj formam uma base para a abstração de composição.

A abstração de agregação, Figura 2, é mapeada através de várias relações, por exemplo as relações: tipo_atributo, vincula_atributo_objeto e a relação caract_texto, em particular o item CodObj é referenciado pelas relações: vincula_atributo_objeto e caract_texto, sendo as duas últimas referenciadas pela relação tipo_atributo. Desta forma, todos os construtores semânticos do modelo são mapeados para o modelo relacional através da relação objeto e as demais relações são

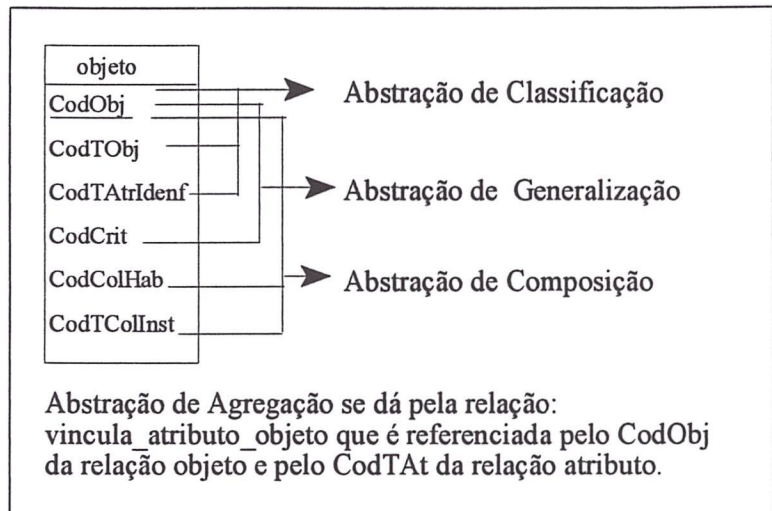


Figura 1 - Relação Objeto

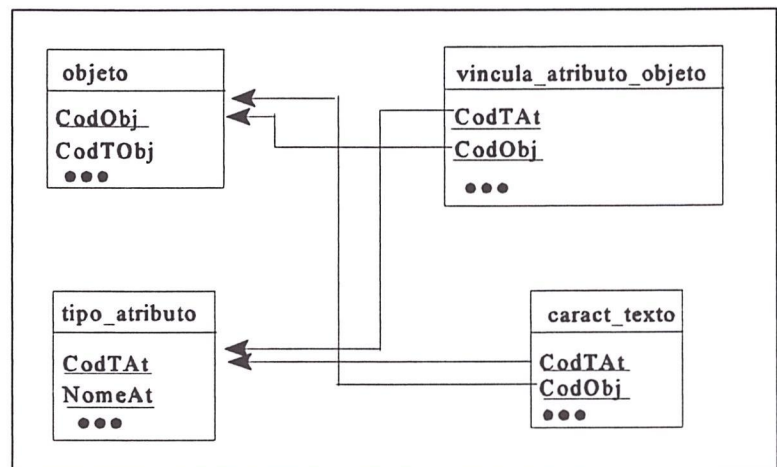


Figura 2 - Relações da Abstração de Agregação

derivadas dessa. A Figura 2 mostra algumas relações derivadas. A Figura 3 ilustra o conjunto das principais relações utilizadas na implementação do Editor de Esquemas - E²SIRIUS, bem como suas chaves primárias (sublinhadas) e as referências de chaves estrangeiras (indicadas por setas).

Após a criação de um repositório de dados, o passo seguinte foi a elaboração de uma interface que, como a base, engloba todos os construtores semânticos do modelo.

O próximo capítulo, descreve algumas características da fase de desenvolvimento de projeto do editor E²SIRIUS, assim como algumas de suas primitivas relevantes.

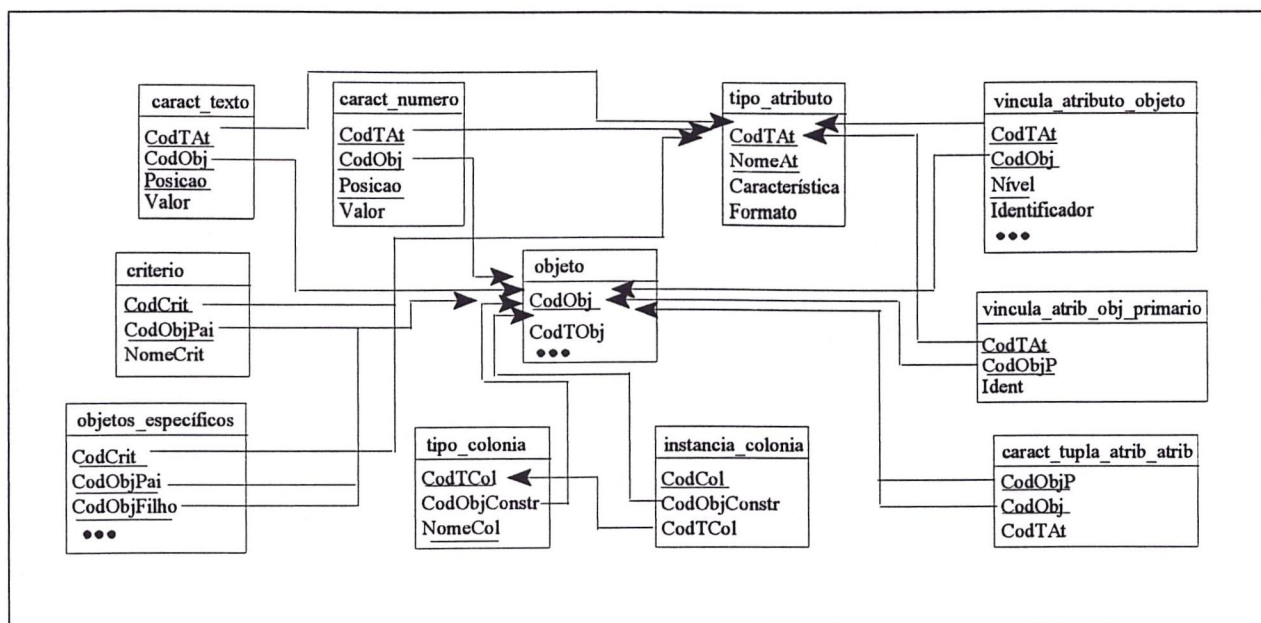


Figura 3 - Esquema Relacional para SIRIUS

Capítulo 3

Implementação

O E²SIRIUS foi implementado de forma que os dados são digitados em campos específicos de certos formulários. Por exemplo, no interador Objeto, ilustrado na Figura 4, o usuário digita qual o nome do objeto no campo apropriado. Uma vez fornecido o nome de um objeto, o sistema gera o código para ele e o armazena, juntamente com seu nome, nas relações apropriadas. É importante citar que o Editor também faz o controle das restrições de integridade, evitando desse modo, entre outras coisas, que o usuário armazene dados inválidos e/ou duplicados na base, assim

Objeto

Abstrato Detalhe Relacionamento

Atributos de Classificação

NOME	CARACT.	NÍVEL	IDENT.	DEFAULT	ORGANIZAÇÃO
------	---------	-------	--------	---------	-------------

At. Classif
Edita Valor

Objeto

Tipo do Objeto Tipo da Colônia

Meta Tipo Obj

Nome do Objeto

Define Colônia

+ Objeto - Objeto

Topo

+ Atributo - Atributo

Abstrato

Classificar Generalizar Constringir

Detalhe

Instanciar Especializar Habitar

Atributos de Instanciación

ATRIBUTO	SUB_ATRIBUTO	CARACT.	VALOR
----------	--------------	---------	-------

At. Instanc.
Edita Valor
Def. Valor
Def. Atr. Extra

Fechar

Figura 4 - Formulário Objeto

para cada formulário que o usuário irá utilizar para armazenar informações existe um controle, ou seja, as restrições de integridade e controle de consistência foram levados em consideração tanto em nível de aplicativo como para a criação do repositório de dados. Com isso implementou-se os construtores semânticos do modelo SIRIUS.

Desta forma todas as abstrações suportadas pelo modelo estão presentes no editor de esquemas. Estas foram agrupadas em subconjuntos de formulários que abrangem apenas uma das abstrações de cada vez. Em outras palavras, os subconjuntos de formulários foram agrupados de modo a obter-se uma coleção mínima de formulários.

Em particular, o E²SIRIUS conta com um pequeno conjunto de telas (onde não existe interação entre o usuário e a base) e um conjunto considerável de formulários (onde existe interação entre o usuário e a base). É através dos formulários que o usuário pode armazenar e/ou consultar e/ou alterar seus dados na base.

Na implementação do editor a ferramenta de desenvolvimento utilizada foi o C++Builder Cliente/Server Suite versão 3.0.

O primeiro protótipo de E²SIRIUS desenvolvido, teve aproximadamente 6 meses de duração. Esta implementação compõe-se de aproximadamente 24 *units*, englobando 94 rotinas. Ao formulário Objeto (Figura 4) está associada a *unit* U_Objeto.cpp contendo o código do formulário Objeto, elemento central na criação do editor. Devido a sua importância este código encontra-se em Anexo.

Na implementação em questão tão importante quanto as rotinas são os códigos elaborados para os interadores (em particular os botões) e as ações associadas aos eventos.

3.1 Rotinas de E²SIRIUS

Esta seção ilustra as rotinas mais relevantes do editor. Para cada rotina exibi-se: o nome, o arquivo que a contém, a funcionalidade, o(s) parâmetro(s) de entrada e retorno (quando existir(em)).

Rotina:	Inserir_Objeto
Arquivo:	U_Objeto.cpp
Funcionalidade:	Insere informações associadas a um objeto na relação objeto
Parâmetro(s) de entrada:	código do objeto, código do tipo do objeto, nome do objeto, código da colônia que habita, código do tipo da colônia das instâncias
Retorno:	não há retorno

Rotina:	Inserir_Valor_Texto
Arquivo:	U_Objeto.cpp
Funcionalidade:	Insere o valor de um atributo com característica texto, ou o nome de um objeto, na relação caract_texto
Parâmetro(s) de entrada:	código do atributo, código do objeto, posição do valor, valor
Retorno:	não há retorno

Rotina:	Inserir_Vinculo
Arquivo:	U_Objeto.cpp
Funcionalidade:	Insere o vínculo entre objeto e atributo na relação vincula_atributo_objeto
Parâmetro(s) de entrada:	código do atributo, código do objeto, nível de classificação / instanciação do atributo, identificador (S/N), limite inferior, limite superior, organização, definido (S/N), <i>default</i> (S/N), Fixo
Retorno:	não há retorno

Rotina:	Buscar_Obj_Por_Nome
Arquivo:	U_Objeto.cpp

Funcionalidade:	Retorna código do objeto, buscando em colônia aberta
Parâmetro(s) de entrada:	nome do objeto
Retorno:	código do objeto

Rotina:	Buscar_Obj_Por_Codigo
Arquivo:	U_Objeto.cpp
Funcionalidade:	Retorna o nome do objeto a partir de seu código
Parâmetro(s) de entrada:	código do objeto
Retorno:	nome do objeto

Rotina:	Verifica_Tipo_Objeto
Arquivo:	U_Objeto.cpp
Funcionalidade:	Verifica se o objeto a ser removido é tipo, ou seja, se possui instâncias
Parâmetro(s) de entrada:	código do objeto
Retorno:	<i>true</i> se o objeto é tipo ou <i>false</i> se o objeto não é tipo

Rotina:	Buscar_Tipo_Obj
Arquivo:	U_Objeto.cpp
Funcionalidade:	Busca o tipo de um objeto
Parâmetro(s) de entrada:	código do objeto
Retorno:	código do tipo do objeto

Rotina:	Buscar_Ident_Nivel_Especifico
Arquivo:	U_Objeto.cpp
Funcionalidade:	Busca o código do atributo identificador do objeto
Parâmetro(s) de entrada:	código do objeto, nível de classificação / instanciação
Retorno:	código do atributo identificador

Rotina:	Listar_Objeto
Arquivo:	U_Objeto.cpp
Funcionalidade:	Lista os objetos de um determinado tipo num componente
Parâmetro(s) de entrada:	componente (onde são listados os objetos) e nome do tipo de objeto
Retorno:	não há retorno

Rotina:	Inserir_Vinculos_Fixos
Arquivo:	U_Objeto.cpp
Funcionalidade:	Insere os vínculos fixos do objeto pai ao objeto filho
Parâmetro(s) de entrada:	código do objeto filho, código o objeto pai
Retorno:	não há retorno

Rotina:	Remove_Objeto
Arquivo:	U_Confirma_Remocao_Obj.cpp
Funcionalidade:	Remove da base o objeto corrente, passando como parâmetro o código do objeto que será removido
Parâmetro(s) de entrada:	código do objeto
Retorno:	não há retorno

Rotina:	Lista_Atributos
Arquivo:	U_Ocao.cpp
Funcionalidade:	Lista todos os atributos da base
Parâmetro(s) de entrada:	o componente (onde são listados os atributos)
Retorno:	não há retorno

Rotina:	Verifica_Vinculo_Duplicado
Arquivo:	U_Ocao.cpp
Funcionalidade:	Verifica se já existe um vínculo entre o objeto e o atributo

Parâmetro(s) de entrada:	código o objeto e do atributo
Retorno:	<i>true</i> se já existe vínculo entre o objeto e o atributo <i>false</i> caso contrário

Rotina:	Buscar_ObjPrimario
Arquivo:	U_Vinculo.cpp
Funcionalidade:	Buscar o objeto primário
Parâmetro(s) de entrada:	código do atributo
Retorno:	código do objeto primário vinculado a um atributo

Rotina:	Buscar_Organizacao
Arquivo:	U_Vinculo.cpp
Funcionalidade:	Buscar qual é a organização de um atributo vinculado a um objeto
Parâmetro(s) de entrada:	código do atributo e do objeto
Retorno:	carácter que representa a organização associada ao atributo

Rotina:	Inserir_Atributo
Arquivo:	U_Atributo.cpp
Funcionalidade:	Insere dados do atributo na relação Tipo_atributo
Parâmetro(s) de entrada:	código, nome, característica e formato do atributo
Retorno:	não há retorno

Rotina:	Lista_Atributos_Especificos
Arquivo:	U_Atributo.cpp
Funcionalidade:	Lista os atributos de acordo com sua característica
Parâmetro(s) de entrada:	componente e o nome da característica do atributo
Retorno:	não há retorno

Rotina:	Inserir_Vinculo_ObjPrimario
Arquivo:	U_Atributo.cpp
Funcionalidade:	Efetuar o vínculo entre um atributo e o objeto primário de um sub atributo com característica tupla ou atributo de atributo
Parâmetro(s) de entrada:	código do atributo, código do objeto e informação se o atributo é um identificador ou não (tal informação é utilizada em atributos com característica atributo de atributo)
Retorno:	não há retorno

Rotina:	Buscar_Atrib_Ident
Arquivo:	U_Remover_Atributo.cpp
Funcionalidade:	Buscar atributo identificador
Parâmetro(s) de entrada:	código do objeto e do atributo
Retorno:	<i>true</i> e o atributo for identificador e <i>false</i> caso contrário

Rotina:	Remove_Vinculo_Atributo
Arquivo:	U_Confirma_Remocao_Atributo.cpp
Funcionalidade:	Remove o vínculo do atributo com o objeto corrente
Parâmetro(s) de entrada:	código do atributo a ser removido e o código do objeto que esta vinculado a este atributo
Retorno:	não há retorno

Rotina:	Remove_Atributo
Arquivo:	U_Confirma_Remocao_Atributo.cpp
Funcionalidade:	Remover um atributo da base
Parâmetro(s) de entrada:	código do atributo que será removido
Retorno:	não há retorno

Rotina:	Vincula_SubAtr_ObjPrim
Arquivo:	U_Sub_Atrib.cpp
Funcionalidade:	Efetua o vínculo entre o sub atributo e o(s) objeto(s) primário(s) correspondente(s) de um sub atributo com característica tupla ou atributo de atributo
Parâmetro(s) de entrada:	código do atributo, código do sub atributo e informação se o atributo é um identificador ou não (tal informação é utilizada em atributos com característica atributo de atributo)
Retorno:	não há retorno

Rotina:	Verifica_Duplicidade_Atrib_Atrib
Arquivo:	U_Sub_Atrib.cpp
Funcionalidade:	Verifica a duplicidade entre um atributo e seu(s) sub atributo(s)
Parâmetro(s) de entrada:	nome do atributo e do sub atributo
Retorno:	<i>true</i> quando existir um atributo duplicado e <i>false</i> caso contrário

Rotina:	Remover_Sub_Atributo
Arquivo:	U_Confirma_Remocao_Sub_Atrib.cpp
Funcionalidade:	Remove um sub atributo da base, removendo primeiramente o objeto primário da relação Vincula_atrib_obj_primario
Parâmetro(s) de entrada:	código do atributo e do sub atributo (vinculado ao atributo)
Retorno:	não há retorno

Rotina:	Listar_Atrib_Instancia
Arquivo:	U_Edita_Valor.cpp
Funcionalidade:	Lista os atributos de instanciação de um objeto
Parâmetro(s) de entrada:	componente e nome do objeto
Retorno:	não há retorno

Rotina:	Buscar_Valor
Arquivo:	U_Edit_Valor.cpp
Funcionalidade:	Busca o valor de um atributo vinculado a um objeto, em uma determinada posição
Parâmetro(s) de entrada:	código do objeto, código do atributo, posição e característica do atributo
Retorno:	valor do atributo

Rotina:	Modificar_Valor_Texto
Arquivo:	U_Edit_Valor.cpp
Funcionalidade:	Altera o valor, numa determinada posição, de um atributo com característica texto, vinculado a um objeto
Parâmetro(s) de entrada:	códigos do objeto e do atributo, posição e novo valor do atributo
Retorno:	não há retorno

Rotina:	Buscar_Ultima_Posicao
Arquivo:	U_Edit_Valor.cpp
Funcionalidade:	Busca a última posição alocada para um atributo multivalorado (lista, conjunto ou vetor) e o número de elementos já inseridos
Parâmetro(s) de entrada:	código do objeto e código do atributo
Retorno:	última posição e número de elementos, armazenados na estrutura Multivalorado

Rotina:	Listar_Atrib_Pai
Arquivo:	U_Def_Valor.cpp
Funcionalidade:	Lista os atributos não fixos vinculados ao objeto pai e não vinculados ao objeto filho
Parâmetro(s) de entrada:	componente onde serão listados os atributos, nome do objeto pai e nome do objeto filho

Retorno:	não há retorno
----------	----------------

Rotina:	Herdar_atributos
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Vincula todos os atributos do objeto genérico ao objeto específico
Parâmetro(s) de entrada:	códigos dos objetos genérico e específico
Retorno:	não há retorno

Rotina:	Inserir_Obj_Especifico
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Insere os dados do objeto específico na relação objetos_especificos
Parâmetro(s) de entrada:	código do critério, códigos dos objetos genérico e específico, predicado e valores de inicialização
Retorno:	não há retorno

Rotina:	Atrib_Valido_Para_Criterio
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Verifica se um atributo pode ser critério de especialização ou seja, se está vinculado ao objeto genérico, com nível 1 e sem valor <i>default</i>
Parâmetro(s) de entrada:	nome do atributo a ser utilizado como critério e código do objeto genérico
Retorno:	true se o atributo é válido e false caso contrário

Rotina:	Inserir_Criterio
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Insere dados do critério na relação critério

Parâmetro(s) de entrada:	código e nome do critério, código do objeto genérico, dados das restrições de sobreposição, participação e categorização
Retorno:	não há retorno

Rotina:	Listar_Criterios
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Lista os critérios segundo os quais um objeto generalização foi especializado, de acordo com o tipo de generalização (Especialização Restrita ou Categorização). Cada critério corresponde a um tipo diferente de especialização
Parâmetro(s) de entrada:	componente onde os critérios serão listados, código do objeto generalização e restrição de categorização
Retorno:	não há retorno

Rotina:	Buscar_Restricoes
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Busca as opções de restrições, para um objeto generalização, segundo um determinado critério
Parâmetro(s) de entrada:	código do critério e código do objeto generalização
Retorno:	restrições de Sobreposição e Participação na estrutura Restrições

Rotina:	Buscar_Pred_Inic
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Busca o predicado e a inicialização definidos para um objeto específico, segundo um determinado critério
Parâmetro(s) de entrada:	código do objeto específico e código do critério
Retorno:	predicado e inicialização armazenados na estrutura Pred_Inic

Rotina:	Remover_Obj_Especifico
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Remove os dados do objeto especifico na relação objetos_especificos
Parâmetro(s) de entrada:	código do objeto específico
Retorno:	não há retorno

Rotina:	Possui_Especificos
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Verifica se um objeto está especializado em algum objeto específico, segundo um determinado critério
Parâmetro(s) de entrada:	código do objeto e código do critério
Retorno:	<i>true</i> se o objeto estiver especializado em algum objeto específico segundo um determinado critério e <i>false</i> caso contrário

Rotina:	Busca_Obj_Generico
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Retorna o objeto generalização do nível mais alto da hierarquia de generalização. Este objeto será o tipo das instâncias de um objeto específico
Parâmetro(s) de entrada:	código do objeto (específico ou não)
Retorno:	código do objeto generalização do nível mais alto da hierarquia de generalização

Rotina:	Buscar_TColonia_Por_Nome
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Retorna o código do tipo da colônia a partir de seu nome
Parâmetro(s) de entrada:	nome do tipo da colônia
Retorno:	código do tipo da colônia

Rotina:	Inserir_Tipo_Colonia
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Insere informações associadas a um tipo de colonia na relação tipo_colonia
Parâmetro(s) de entrada:	código do tipo de colônia, código do objeto que constrixe e nome do tipo de colônia
Retorno:	não há retorno

Rotina:	Buscar_TCol_Constringe
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Retorna os códigos dos tipos de colônia que o objeto constrixe
Parâmetro(s) de entrada:	código do objeto
Retorno:	lista com os códigos dos tipos de colônia constritos pelo objeto

Rotina:	Colonia_Aberta
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Retorna a colônia aberta de acordo com um determinado tipo de colônia
Parâmetro(s) de entrada:	código do tipo de colônia (será buscada uma colônia aberta deste tipo)
Retorno:	código da colônia aberta encontrada

Rotina:	Buscar_Colonia_Constrita
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Busca o código da colônia, de um determinado tipo, constrita por um objeto específico
Parâmetro(s) de entrada:	código do tipo da colônia e código do objeto que constrixe
Retorno:	código da colônia

Rotina:	Inserir_Instanceia_Colonia
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Insere as informações de uma colônia na relação instancia_colonia
Parâmetro(s) de entrada:	código da colônia, código do objeto que a constringe e código do tipo da colônia
Retorno:	não há retorno

Rotina:	Fechar_Colonia
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Fecha a colônia de um determinado tipo (este procedimento retira colônia da relação de colônias abertas)
Parâmetro(s) de entrada:	código do tipo da colônia a ser fechada
Retorno:	não há retorno

Rotina:	Buscar_TColonia_Por_Codigo
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Busca o nome do tipo de colônia a partir de seu código
Parâmetro(s) de entrada:	código do tipo de colônia
Retorno:	nome do tipo de colônia

Rotina:	Buscar_TColonia_Instanceias
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Busca o código do tipo da colônia onde habitam as instâncias de um determinado objeto
Parâmetro(s) de entrada:	código do objeto
Retorno:	código do tipo da colônia onde habitam as instância do objeto

Rotina:	Buscar_Colonia_Habita
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Busca o código da colônia onde um determinado objeto habita
Parâmetro(s) de entrada:	código do objeto
Retorno:	código da colônia onde o objeto habita

Rotina:	Existe_Instancia
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Verifica se existe alguma colônia instanciada de um tipo de colônia específico
Parâmetro(s) de entrada:	código do tipo de colônia
Retorno:	lista dos códigos das instâncias (colônias) do tipo_colônia

Rotina:	Colonia_Vazia
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Verifica se uma colônia está vazia
Parâmetro(s) de entrada:	código da colônia
Retorno:	<i>true</i> se a colônia está vazia e <i>false</i> caso contrário

Rotina:	Abrir_Colonia
Arquivo:	U_Colonias_Abertas.cpp
Funcionalidade:	Insere uma colônia na relação utilizada para armazenar as colônias abertas
Parâmetro(s) de entrada:	código da colônia e código do tipo da colônia
Retorno:	não há retorno

3.2 Interadores

Este tópico exhibe alguns dos interadores implementados. Em particular os botões mais relevantes. Desta forma cada interador será apresentado por meio do seu nome, da sua funcionalidade e do contexto que este está inserido, ou seja, a que arquivo este pertence.

Botão:	Constringir
Arquivo:	U_Objeto.cpp
Funcionalidade:	Abre ou cria uma colônia constringida pelo objeto corrente, sendo o que o tipo desta colônia é constringido pelo tipo do objeto corrente. Se já existe uma colônia do mesmo tipo aberta, esta colônia é fechada e uma nova colônia é aberta (Obs: O modelo não permite a existência de duas colônias do mesmo tipo abertas numa aplicação monousuário)

Botão:	+ Atributo (Adiciona Atributo)
Arquivo:	U_Objeto.cpp
Funcionalidade:	Possibilita a definição de um novo atributo ou vínculo de um atributo já existente ao objeto corrente

Botão:	- Atributo (Remove Atributo)
Arquivo:	U_Objeto.cpp
Funcionalidade:	Possibilita a remoção de atributos da base ou o vínculo de atributos com o objeto corrente

Botão:	+ Objeto (Inserir Objeto)
Arquivo:	U_Objeto.cpp
Funcionalidade:	Insere um novo objeto e/ou cria uma instância de um objeto definido anteriormente

Botão:	- Objeto (Remove Objeto)
Arquivo:	U_Objeto.cpp
Funcionalidade:	Remove objeto corrente da base, se este não for tipo, se não restringir colônia habitada e se não for objeto genérico

Botão:	Especializar
Arquivo:	U_Objeto.cpp
Funcionalidade:	Verifica se o objeto pode ser especializado e chama o formulário para determinação do tipo de especialização: Especialização Restrita ou Categorização

Botão:	Instanciar
Arquivo:	U_Objeto.cpp
Funcionalidade:	Verifica se o objeto corrente pode ser instanciado

Botão:	Classificar
Arquivo:	U_Objeto.cpp
Funcionalidade:	Classifica o objeto corrente

Botão:	Define Colônia
Arquivo:	U_Objeto.cpp
Funcionalidade:	Chama o formulário Tipo_Colonia (Define Tipo Colônia) para definição de tipo de colônia

Botão:	Habitar
Arquivo:	U_Objeto.cpp
Funcionalidade:	Chama o formulário Habitar (Habita), para mostrar a colônia onde cada objeto da base habita

Botão:	Generalizar
Arquivo:	U_Objeto.cpp
Funcionalidade:	Chama o formulário Generalização (Generalização), para mostrar informações da hierarquia de generalização, de acordo com o tipo do objeto corrente

Botão:	Vincula
Arquivo:	U_Opcao.cpp
Funcionalidade:	Verifica se o objeto a ser vinculado ao atributo foi selecionado, em caso afirmativo chama o formulário Vinculo que irá fazer o vínculo entre objeto e o atributo

Botão:	Salvar
Arquivo:	U_Vinculo.cpp
Funcionalidade:	Salvar informações definidas no vínculo entre o objeto corrente e um atributo

Botão:	Salvar
Arquivo:	U_Atributo.cpp
Funcionalidade:	Salvar valores definidos para os atributos de acordo com a sua característica

Botão:	Remove Atributo
Arquivo:	U_Remover_Atributo.cpp
Funcionalidade:	Efetua testes para remoção de atributo, chama o formulário que irá de fato efetuar a remoção e por fim atualiza os <i>grids</i> do formulário Objeto

Botão:	Sim
--------	-----

Arquivo:	U_Confirma_Remocao_Atributo.cpp
Funcionalidade:	Confirma a remoção do atributo, após a confirmação remove apenas o vínculo do atributo com o objeto corrente ou remove o atributo da base.

Botão:	+ Valor (Adiciona Valor)
Arquivo:	U_Edita_Valor_Classif.cpp
Funcionalidade:	Insere um valor para um atributo ou sub atributo, ou seja, inserção de um valor para atributo ou sub atributo de classificação - valor <i>default</i> .

Botão:	- Valor (Remove Valor)
Arquivo:	U_Edita_Valor_Classif.cpp
Funcionalidade:	Remove um valor atribuído ao atributo corrente. OBS: a posição ocupada pelo valor removido não será mais utilizada - se um novo valor for inserido, uma nova posição será alocada

Botão:	+ Valor (Adiciona Valor)
Arquivo:	U_Edita_Valor.cpp
Funcionalidade:	Insere um valor para um atributo ou sub atributo

Botão:	- Valor (Remove Valor)
Arquivo:	U_Edita_Valor.cpp
Funcionalidade:	Remove um valor atribuído ao atributo corrente. OBS: a posição ocupada pelo valor removido não será mais utilizada - se um novo valor for inserido, uma nova posição será alocada

Botão:	+ Valor (Adiciona Valor)
Arquivo:	U_Def_Valor.cpp
Funcionalidade:	Vincula um atributo selecionado no campo nome atributo (atributo do objeto pai) ao objeto corrente (objeto filho) e insere um valor para este atributo (associado ao objeto corrente)

Botão:	Salvar
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Insere na base as informações associadas à abstração de generalização quando objetos específicos são criados

Botão:	- Obj. Espec. (Remove Objeto Específico)
Arquivo:	U_Especializacao.cpp
Funcionalidade:	Remove o objeto específico corrente da hierarquia de generalização

Botão:	+ Tipo Colônia (Insere Tipo Colônia)
Arquivo:	U_Tipo_Colonia.cpp
Funcionalidade:	Insere um novo tipo de colônia

3.3 Ação(ões) associada(s) ao evento

Existem eventos, por exemplo: evento *Click*, evento *DropDown* entre outros, que possuem ações associadas, ou seja, ativando-se um evento uma série de comandos serão executados. A implementação foi feita de forma que alguns eventos irão disparar ações importantes para o bom funcionamento do editor. Os eventos mais relevantes são destacados e ilustrados a seguir.

Evento:	<i>DropDown</i> para o campo nome do objeto
Arquivo:	U_Objeto.cpp
Ação(ões) associada(s):	Lista os nomes dos objetos do tipo corrente, que habitam uma colônia aberta (cujo tipo é constringido pelo tipo de objeto corrente)

Evento:	<i>DropDown</i> para o campo tipo de colônia
Arquivo:	U_Objeto.cpp
Ação(ões) associada(s):	Lista os tipos de colônias

Evento:	<i>DropDown</i> para o campo tipo de objeto
Arquivo:	U_Objeto.cpp
Ação(ões) associada(s):	Lista os tipos de objetos no campo tipo objeto

Evento:	<i>Click</i> para o campo Classif./Instanc.
Arquivo:	U_Vinculo.cpp
Ação(ões) associada(s):	Limpa todos os campos do formulário vínculo. Habilita e/ou desabilita campos (do formulário Vínculo) apropriadamente de acordo com a escolha (atributo de classificação ou atributo de instânciação) feita pelo usuário

Evento:	<i>Click</i> do campo Atrib. indefinido
Arquivo:	U_Vinculo.cpp

Ação(ões) associda(s):	Habilita e/ou desabilita campos (do formulário Vinculo) apropriadamente de acordo com a escolha (atributo indefinido ou atributo não indefinido) feita pelo usuário
------------------------	---

Evento:	<i>Click</i> do campo Identificador
Arquivo:	U_Vinculo.cpp
Ação(ões) associda(s):	Habilita e/ou desabilita campos (do formulário Vinculo) apropriadamente de acordo com a escolha (atributo identificador ou não) feita pelo usuário

Evento:	<i>Click</i> do campo Valor nulo
Arquivo:	U_Vinculo.cpp
Ação(ões) associda(s):	Habilita e/ou desabilita campos (do formulário Vinculo) apropriadamente de acordo com a escolha (atributo com valor nulo ou não) feita pelo usuário

Evento:	<i>DropDown</i> para o campo nome do atributo
Arquivo:	U_Vinculo.cpp
Ação(ões) associda(s):	Lista atributos

Evento:	<i>DropDown</i> para o campo nome do atributo
Arquivo:	U_Atributo.cpp
Ação(ões) associda(s):	Lista atributos de acordo com a característica do atributo

Evento:	<i>Click</i> para o campo nome do atributo
Arquivo:	U_Atributo.cpp

Ação(ões) associda(s):	Se um item da lista foi selecionado então ativa os botões adicionar/remover sub atributo, busca o código do atributo a partir o seu nome e atualiza o <i>grid</i> classificação após serem feitos os vínculos necessários Senão desativa os botões adicionar/remover sub atributo.
------------------------	---

Evento:	<i>DropDown</i> para o campo nome do atributo
Arquivo:	U_Remover_Atributo.cpp
Ação(ões) associda(s):	Verifica se alguma opção de remoção foi escolhida. Caso a opção retirar vínculo com o objeto corrente foi selecionada então busca o código do objeto a partir do seu nome e chamada a função que irá listar apenas os atributos vinculados ao objeto corrente. Senão lista todos os atributos da base.

Evento:	<i>DropDown</i> para o campo nome do atributo
Arquivo:	U_Edita_Valor_Classif.cpp
Ação(ões) associda(s):	Lista os atributos de classificação, vinculados ao objeto corrente, que possuem valores <i>default</i>

Evento:	<i>DropDown</i> para o campo nome do atributo
Arquivo:	U_Edita_Valor.cpp
Ação(ões) associda(s):	Lista os atributos de instanciação do objeto

Evento:	<i>DropDown</i> para o campo nome de atributo
Arquivo:	U_Def_Valor.cpp
Ação(ões) associda(s):	Lista os atributos não fixos vinculados ao tipo do objeto corrente (objeto pai) e que não foram vinculados ao objeto corrente (objeto filho)

Evento:	<i>DropDown</i> do campo Nome do Objeto (Objeto generalização)
Arquivo:	U_Especializacao.cpp
Ação(ões) associda(s):	Lista os objetos que podem ser especializados

Evento:	<i>Click</i> (escolha) do campo Nome do Objeto (Objeto generalização)
Arquivo:	U_Especializacao.cpp
Ação(ões) associda(s):	Limpa os campos das restrições de Sobreposição e Participação. Limpa o campo do nome do objeto específico. Limpa o campo do critério de especialização. Limpa o campo do predicado. Limpa o campo dos valores de inicialização. Verifica se algum item (objeto generalização) foi selecionado. Se item selecionado então busca o código do objeto generalização, busca o código e o nome do tipo de colônia onde habitam as instâncias do objeto generalização - os objetos específicos vão habitar colônia do mesmo tipo e preenche os campos de Tipo do Objeto e Tipo de Colônia do objeto específico com os mesmo dados do objeto generalização

Evento:	<i>DropDown</i> do campo nome do objeto específico
Arquivo:	U_Especializacao.cpp
Ação(ões) associda(s):	Lista os objetos específicos já definidos a partir do objeto generalização e do critério corrente

Evento:	<i>DropDown</i> para o campo critério
Arquivo:	U_Especializacao.cpp
Ação(ões) associda(s):	Lista todos os critérios segundo os quais o objeto generalização corrente já foi especializado, de acordo com o tipo de generalização (Especialização Restrita ou Categorização)

Evento:	<i>Click</i> (escolha) para o campo critério
---------	--

Arquivo:	U_Especializacao.cpp
Ação(ões) associda(s):	Limpa os campos de restrições. Limpa o campo de nome do objeto específico. Limpa o campo do predicado. Limpa o campo da inicialização. Verifica se um item (critério) foi selecionado. Se item selecionado então busca as restrições definidas para o objeto generalização e o critério corrente e apresenta na tela as restrições armazenadas na base.

Evento:	<i>Click</i> (escolha) para o campo de nome do objeto específico
Arquivo:	U_Especializacao.cpp
Ação(ões) associda(s):	Limpa o campo do predicado. Limpa o campo da inicialização. Verifica se um item (objeto específico) foi selecionado. Se item selecionado então busca o código do objeto específico, busca o código do critério, busca e apresenta na tela o predicado e a inicialização definidos para o objeto específico segundo o critério corrente.

Evento:	<i>DropDown</i> para o campo nome do objeto que restringe
Arquivo:	U_Tipo_Colonia.cpp
Ação(ões) associda(s):	Lista os nomes dos objetos do tipo corrente, que habitam uma colônia aberta

Evento:	<i>DropDown</i> do campo Nome do Tipo da Colônia
Arquivo:	U_Tipo_Colonia.cpp
Ação(ões) associda(s):	Lista os Tipos de Colônia existentes

Evento:	<i>Click</i> (escolha de um tipo de colônia) do campo Nome do Tipo de Colônia
Arquivo:	U_Tipo_Colonia.cpp

Ação(ões) associada(s):	Verifica se algum item foi selecionado. Se item selecionado então mostra no campo Objeto que Constringe o nome do objeto que constringe o tipo de colônia escolhido
-------------------------	---

Capítulo 4

Considerações finais

A implementação do Editor de Esquemas levou em consideração três aspectos principais: o suporte às abstrações integrada aos demais conceitos do modelo; ser um meio de facilitar a construção de modelagens feitas com um modelo altamente complexo; e constituir-se em uma ferramenta de suporte ao desenvolvimento do próprio gerenciador SIRIUS/GO. De maneira distinta dos demais editores de esquemas e linguagens de consulta e manipulação de gerenciadores de dados, este Editor não tem conjuntos de comandos/formulários diferenciados para uma parte de acesso à definição do esquema (equivalente a uma DDL - *Data Definition Language*) e outra para a manipulação de dados (equivalente a uma DML - *Data Manipulation Language*). Isso porque o modelo de dados suporta a abstração de classificação, permitindo que tipo de objeto seja um objeto com as mesmas propriedades, restrições e suporte do gerenciador de todos os demais objetos.

É importante notar, que a tecnologia atual de implementação de aplicativos, utilizando o modelo relacional e as ferramentas de desenvolvimento rápido de aplicativos (RAD - *Rapid Application Development*), não dão suporte para a abstração de classificação. No entanto, esta abstração pode ser bem suportada em modelos de objetos, e pelas ferramentas de projeto e desenvolvimento que os suportem. O tratamento uniforme dado a objetos que são tipo e a instâncias, é um recurso poderoso, que pode vir a aumentar a flexibilidade das ferramentas que venham a utilizá-lo. Nesse sentido, o Editor apresentado fornece uma boa ilustração de como esses conceitos poderão ser implementados em ferramentas de apoio ao desenvolvimento de *software* centrado em bases de dados.

Por outro lado, a complexidade inerente a SIRIUS torna difícil o treinamento de analistas que possam utilizá-lo para modelar sistemas. No desenvolvimento de E²SIRIUS houve a preocupação de ao mesmo tempo ser possível editar-se os esquemas manipulando explicitamente os conceitos formais que levaram à especificação do modelo - resultando numa ferramenta de apoio ao aprendizado do modelo quanto ser possível a edição de esquemas realizada por projetistas já experientes - resultando numa ferramenta que suporta ao mesmo tempo a intuitividade no

desenvolvimento de modelagens, quanto permite a edição eficiente de esquemas.

Por outro lado, quando se empreende a implementação de um gerenciador de dados completamente independente de outros já existentes, o E²SIRIUS constitui-se num ponto problemático de desenvolvimento, pois ao mesmo tempo em que se precisa do gerenciador já pronto para armazenar os dados dos esquemas editados, o gerenciador somente pode ser desenvolvido depois que seja possível armazenar os dados dos esquemas, que definem as estruturas de armazenagem e acesso dos objetos manipulados. Assim, este trabalho procurou quebrar este ciclo, desenvolvendo o Editor armazenando os esquemas editados numa base relacional. Quando o núcleo do gerenciador SIRIUS estiver com recursos suficientes para armazenar todos os aspectos de dados necessários ao armazenamento dos esquemas, a interface de dados será modificada para que os dados editados passem a ser armazenados nesse gerenciador. A presente implementação já levou em conta essa mudança futura, e isso aumenta a flexibilidade do Editor, que pode ser facilmente configurado para suportar a construção de esquemas para outros modelos de dados que suportem subconjuntos dos conceitos suportados por SIRIUS, como é o caso da maioria dos modelos orientados a objetos atuais.

Anexo

Código do Formulário Objeto

```
//-----
//  Arquivo: U_Objeto.cpp
//  Autores: Myrian Renata Barros Araujo
//           Elaine Machado Parros
//  Orientador: Caetano Traina Júnior
//  Data de Criação: 01 de setembro de 1997
//  Data da Última Atualização: 08 de fevereiro de 1999
//  Versão: 1.0
//  Rotinas:
//      void Inserir_Objeto(int, int, AnsiString, int, int);
//      void Inserir_Valor_Texto(int, int, int, AnsiString);
//      void Inserir_Valor_Número(int, int, int, AnsiString);
//      void Inserir_Vinculo(int, int, int, AnsiString, int, int,
//                          AnsiString, AnsiString, AnsiString, int);
//      int Buscar_Obj_Por_Nome(AnsiString);
//      AnsiString Buscar_Obj_Por_Codigo(int);
//      int Buscar_Ident_Nivel_Especifico(int, int);
//      void Listar_Objetos(TComboBox *, AnsiString);
//      void Inserir_Vinculos_Fixos(int, int);
//      void Limpa_Tela_Objeto(void);
//      bool Verifica_Tipo_Objeto(int);
//      int Buscar_Tipo_Obj(int);
//      void Seta_Valor_Grid(void);
//      TQuery *Buscar_Sub_Atributos(int);
//      void Limpa_Grid_Instanceia(void);
//      void Listar_Tipos_Obj(TComboBox *);
//
//  Modificações Futuras:
//      Data:
//      Autor:
//      Rotina:
//      Motivo:
//      Nova versão:
//-----
#include <vcl\vcl.h>
#include <vcl\dbgrids.hpp>
#pragma hdrstop

#include <stdlib.h>
#include "U_Objeto.h"
#include "U_Atributo.h"
#include "U_ModuloDado.h"
#include "U_Edita_Valor.h"
#include "U_Opcao.h"
#include "U_Vinculo.h"
#include "U_Edita_Valor_Classif.h"
#include "U_Def_Valor.h"
#include "U_Remover_Atributo.h"
#include "U_Confirma_Remocao_Obj.h"
#include "U_Tipo_Colonia.h"
```



```
#include "U_Colonias_Abertas.h"
#include "U_Habitar.h"
#include "U_Especializacao.h"
#include "U_Opcoes_Espec.h"
#include "U_Generalizacao.h"

//-----
#pragma link "Grids"
#pragma resource "*.dfm"
TForm_Objeto *Form_Objeto;

//-----
__fastcall TForm_Objeto::TForm_Objeto(TComponent* Owner)
: TForm(Owner)
{
}
//-----
// Nome do botão: Constringir
// Funcionalidade: Abre ou cria uma colônia constringida pelo objeto corrente,
// sendo o que o tipo desta colônia é constringido pelo tipo do objeto corrente.
// Se já existe uma colônia do mesmo tipo aberta, esta colônia é fechada e uma
// nova colônia é aberta (Obs: O modelo não permite a existência de duas colônias
// do mesmo tipo abertas numa aplicação monousuário).
void __fastcall TForm_Objeto::Button_constringirClick(TObject *Sender)
{
    int objeto, tipo_objeto, colonia, tipo_colonia, colonia_aberta;
    struct Lista *lista_tipo_col;

    // Testa se o campo nome do objeto está selecionado
    if (ComboBox1_nome_objeto->Text == "\0")
    {
        if (MessageBox(0, "Escolha o objeto que vai constringir uma colônia",
            "Informação incompleta", MB_OK) == IDOK)
            return;
    }

    // Busca os códigos do objeto e de seu tipo, a partir de seus nomes
    objeto = Buscar_Obj_Por_Nome (ComboBox1_nome_objeto->Text);
    tipo_objeto = Buscar_Obj_Por_Nome (ComboBox_tipo_obj->Text);

    // Verifica se o tipo do objeto é o Meta Tipo Objeto (código pré-definido = 2)
    // O modelo não permite que objeto cujo tipo é o Meta Tipo Objeto constringa
    // colônia (contexto DRI - Diagrama de Representação de Instância),
    // pois o Meta Tipo Objeto constringe apenas o tipo de colônia Global,
    // que possui uma única instância (pré-definição do modelo e do sistema)
    if (tipo_objeto == 2)
    {
        if (MessageBox(0, "O objeto não pode constringir uma colônia",
            "O objeto é do tipo Meta Tipo Obj", MB_OK) == IDOK)
            return;
    }

    // Busca o tipos de colônias que o tipo do objeto corrente constringe
    // Contexto DHC - Diagrama Hierárquico de Colônia
    lista_tipo_col = Buscar_TCol_Constringe(tipo_objeto);

    // Verifica se a lista de tipos de colônias está vazia
    if (lista_tipo_col == NULL)
    {
        if (MessageBox(0, "O tipo do objeto não constringe tipo de colônia",
            "O objeto não pode constringir uma colônia", MB_OK) == IDOK)
            return;
    }

    // Percorre a lista e trata cada um dos tipos de colônia
```

```

while(lista_tipo_col != NULL)
{
    tipo_colonia = lista_tipo_col->no;

    // Busca o código da colônia constrita pelo objeto corrente
    colonia = Buscar_Colonia_Constrita(tipo_colonia, objeto);

    if (colonia == 0) // Não existe uma colônia constrita pelo objeto
    {
        // Cria uma nova colônia, gera seu código e insere na tabela
        // Instancia_Colonia
        ModuloDado->Gerador->Prepare();
        ModuloDado->Gerador->ExecProc();
        colonia = ModuloDado->Gerador->ParamByName("codigo_Retorno")->AsInteger;
        Inserir_Instance_Colonia(colonia, objeto, tipo_colonia);
        } // fim do if (colonia == 0)

        // Verifica se existe alguma colônia aberta do tipo encontrado
        // Apenas uma colônia de cada tipo pode estar aberta ao mesmo tempo
        if ((colonia_aberta = Colonia_Aberta(tipo_colonia)) != 0) // Existe colônia aberta
        // Fecha a colônia já aberta para abrir a requisitada
        Fechar_Colonia(tipo_colonia);
        // Abrir a nova colônia
        Abrir_Colonia(colonia, tipo_colonia);

        // Busca próximo tipo de colônia na lista
        lista_tipo_col = lista_tipo_col->proximo;

    } // fim do while lista não vazia

    // Atualiza o grid de colônias abertas
    Setar_Colonias_Abertas();

    free(lista_tipo_col); // Libera espaço de lista_tipo_col

    if (MessageBox(0,"A colônia constrita por este objeto foi aberta",
        "Confirmação", MB_OK)==IDOK)
        return;

} // Fim do botão: Constringir

//-----
// Nome do botão: + Atributo (Adiciona Atributo)
// Funcionalidade: Possibilita a definição de um novo atributo ou
// vínculo de um atributo já existente ao objeto corrente
void __fastcall TForm_Objeto::BitBtn_adiciona_atributoClick(TObject *Sender)
{
    // Atualiza o grid de atributos
    ModuloDado->Grid_Atributos->Close();
    ModuloDado->Grid_Atributos->Open();

    // Chama o formulário Opção (Define/Vincula Atributos)
    Form_Opcao->ShowModal();
    Form_Opcao->Button_vincula->Enabled = true;
} // Fim do botão: + Atributo

//-----
// Nome do Botão: - Atributo (Remove Atributo)
// Funcionalidade: Possibilita a remoção de atributos da base ou
// o vínculo de atributos com o objeto corrente
void __fastcall TForm_Objeto::BitBtn_remove_atributoClick(TObject *Sender)

```

```
{
    // Limpa e chama o formulário Remover Atributo
    Form_Remover_Atributo->RadioGroup_remove_atrib->ItemIndex = -1;
    Form_Remover_Atributo->ComboBox1_nome_atributo->Clear();
    Form_Remover_Atributo->ShowModal();
} // Fim do botão: - Atributo

// _____
// Nome do Botão: - Objeto (Remove Objeto)
// Funcionalidade: Remove objeto corrente da base, se este não for tipo, se
// não restringir colônia habitada e se não for objeto genérico
void __fastcall TForm_Objeto::BitBtn_remove_objetoClick(TObject *Sender)
{
    int Cod_obj, tipo_colonia, colonia, tipo_objeto;
    struct Lista *lista_tipo_col; // lista encadeada que armazena os códigos
    // dos tipos de colônias constringidas por um objeto

    // Testa se o campo nome do objeto está preenchido
    if (ComboBox1_nome_objeto->Text == "")
    {
        if (MessageBox(0, "Selecione objeto a ser removido",
            "Informação incompleta", MB_OK) == IDOK)
            return;
    }

    // Busca o código do objeto corrente
    Cod_obj = Buscar_Obj_Por_Nome (ComboBox1_nome_objeto->Text);

    // Verifica se o objeto a ser removido é genérico. Se for, não pode ser
    // removido antes dos específicos serem removidos via aplicativo
    if (Objeto_Generico(Cod_obj))
    {
        if (MessageBox(0, "O objeto está especializado em objetos específicos",
            "Objeto não pode ser removido", MB_OK) == IDOK)
            return;
    }

    // Verifica se o objeto a ser removido é tipo. Se for, não pode ser
    // removido antes de suas instâncias serem removidas via aplicativo
    if (Verifica_Tipo_Objeto(Cod_obj))
    {
        if (MessageBox(0, "O objeto possui instâncias",
            "Objeto não pode ser removido", MB_OK) == IDOK)
            return;
    }

    // Verifica se o objeto a ser removido constringe tipo de colônia
    // (contexto DHC)
    // Se sim, o objeto não pode ser removido antes da remoção do tipo
    // da colônia via aplicativo
    if ((lista_tipo_col = Buscar_TCol_Constringe(Cod_obj)) != NULL)
    {
        if (MessageBox(0, "O objeto constringe tipo de colônia",
            "Objeto não pode ser removido", MB_OK) == IDOK)
            return;
    }

    // Busca o código do tipo do objeto corrente
    tipo_objeto = Buscar_Tipo_Obj(Cod_obj);

    // Busca os tipos de colônia que tipo do objeto a ser removido constringe
    lista_tipo_col = Buscar_TCol_Constringe(tipo_objeto);
```



```
// Percorre a lista e trata cada um dos tipos de colônia
while(lista_tipo_col != NULL) // Se lista não está vazia
{
    tipo_colonia = lista_tipo_col->no;

    // Verifica se o objeto a ser removido constribe alguma colônia do
    // tipo_colonia
    if ((colonia = Buscar_Colonia_Constrita(tipo_colonia, Cod_obj)) != 0)
    {
        // Verifica se a colônia está vazia
        if (!Colonia_Vazia(colonia))
        {
            if (MessageBox(0, "O objeto constribe colônia habitada",
                "Objeto não pode ser removido", MB_OK) == IDOK)
                return;
        }
        else Fechar_Colonia(tipo_colonia); // Fecha a colônia para que possa
            // ser removida
    }

    lista_tipo_col = lista_tipo_col->proximo; // Busca o proximo valor da lista
}

// fim do while lista não vazia

// Chama o Form_Confirma_Remocao_Obj para remover o objeto
Form_Confirma_Remocao_Obj->ShowModal();

// Limpa o nome do objeto
ComboBox1_nome_objeto->Text = "\\0";

// Atualiza os grids de classificação e instanciação
ModuloDado->Grid_Classificacao->Close();
Limpa_Grid_Instanceia();

free(lista_tipo_col); // Libera espaço de lista_tipo_col

// Atualiza o grid de colônias abertas
Setar_Colonias_Abertas();

} // Fim do botão: Remove Objeto

//-----
// Nome do Botão: Especializar
// Funcionalidade: Verifica se o objeto pode ser especializado e chama o
// formulário para determinação do tipo de especialização: Especialização
// Restrita ou Categorização
void __fastcall TForm_Objeto::Button_especializarClick(TObject *Sender)
{
    int codigo_objeto, tipo_col_habita, colonia_aberta;

    // Busca o código do objeto a partir de seu nome
    codigo_objeto = Buscar_Obj_Por_Nome(ComboBox1_nome_objeto->Text);

    // Testa se o campo nome do objeto está preenchido
    if (Form_Objeto->ComboBox1_nome_objeto->Text == "\\0")
    {
        if (MessageBox(0, "O objeto a ser especializado não foi escolhido",
            "Informação incompleta", MB_OK) == IDOK)
            return;
    }
}
```



```

// Verifica se o objeto escolhido pode ser especializado
// Verifica se ele possui atributo identificador
if (Buscar_Ident_Nivel_Especifico(codigo_objeto, 1) == 0)
    if (MessageBox(0, "O objeto escolhido não possui identificador",
        "O objeto não pode ser especializado", MB_OK) == IDOK)
        return;

// Verifica se o objeto escolhido pode ser especializado
// Verifica se ele possui tipo de colônia definido para as instâncias
if (ComboBox1_colonia->Text == "\0")
    if (MessageBox(0, "O objeto escolhido não possui tipo de colônia definido para suas instâncias",
        "O objeto especificado não pode ser especializado", MB_OK) == IDOK)
        return;

// Busca o tipo da colônia onde os objetos específicos vão habitar -
// que é o tipo de colônia onde habitam as instâncias do objeto genérico
tipo_col_habita = Buscar_TColonia_Por_Nome(ComboBox1_colonia->Text);

// Verifica se existe alguma colônia aberta para os objetos específicos
// Se não houver, o objeto não pode ser especializado antes que a colônia
// apropriada seja aberta
if ((colonia_aberta = Colonia_Aberta(tipo_col_habita)) == 0) // Não existe
    // colônia aberta
    if (MessageBox(0, "Não existe colônia aberta para os objetos específicos",
        "O objeto não pode ser especializado", MB_OK) == IDOK)
        return;

// Chama o formulário Opcoes_Espec (Especialização Restrita/Categorização)
Form_Opcoes_Espec->ShowModal();

} // Fim do botão: Especializar

//-----
// Nome do Botão: + Objeto (Inserir Objeto)
// Funcionalidade: Insere um novo objeto e/ou cria uma instância de um objeto
// definido anteriormente
void __fastcall TForm_Objeto::BitBtn1_salvar_Click(TObject *Sender)
{
    int codigo_objeto, tipo_corrente, tipo_objeto, tipo_col_instancias,
        tipo_col_habita, objeto_generico, colonia_aberta, codigo_atributo;
    bool condicao;

    // Testa se o campo nome e o tipo do objeto estão preenchido
    if (Form_Objeto->ComboBox1_nome_objeto->Text == "\0" ||
        Form_Objeto->ComboBox_tipo_obj->ItemIndex == -1)
    {
        if (MessageBox(0, "Certifique-se de que todos os campos estejam preenchidos",
            "Informação incompleta", MB_OK) == IDOK)
            return;
    }

    // Busca o código do tipo de objeto definido para o objeto a ser inserido
    tipo_corrente = Buscar_Obj_Por_Nome(ComboBox_tipo_obj->Text);

    // Busca o tipo da colônia onde o objeto vai habitar
    tipo_col_habita = Buscar_TColonia_Instanceias(tipo_corrente);

    // Verifica se existe alguma colônia (aberta) do tipo encontrado
    // Se não houver, o objeto não pode ser inserido na base antes que a colônia
    // apropriada seja aberta
    if ((colonia_aberta = Colonia_Aberta(tipo_col_habita)) == 0) // Não existe colônia aberta

```

```

    if (MessageBox(0,"Não existe colônia aberta",
        "Informação incompleta", MB_OK)==IDOK)
        return;

// Verifica se já existe outro objeto com o mesmo nome na mesma colônia
if ((codigo_objeto = Buscar_Obj_Por_Nome (ComboBox1_nome_objeto->Text))!=0)
    if (MessageBox(0,"O objeto especificado já existe", "Valor duplicado", MB_OK)==IDOK)
        return;

// Verifica se o objeto a ser instanciado é um objeto específico. Se for,
// busca-se o objeto do nível mais alto da hierarquia de generalização. Este
// objeto será o tipo do objeto instância. O nome do objeto genérico é colocado
// no campo de tipo de objeto na tela.
// Caso contrário, o tipo do objeto instância é o tipo corrente.
// Para inserir vínculos fixos, são utilizados os atributos do objeto específico,
// pois este contém todos os atributos dos níveis superiores da hierarquia
if ((objeto_generico = Busca_Obj_Generico(tipo_corrente)) != tipo_corrente)
{
    Listar_Tipos_Obj(Form_Objeto->ComboBox_tipo_obj);
    for (int i=0; i<=0; i++)
        if(ComboBox_tipo_obj->Items->Strings[i] == (Buscar_Obj_Por_Codigo(objeto_generico)))
        {
            ComboBox_tipo_obj->ItemIndex = i;
            i = -2;
        }
    tipo_objeto = objeto_generico;
}
else tipo_objeto = tipo_corrente;

// Gera código para o objeto a ser inserido na base
ModuloDado->Gerador->Prepare();
ModuloDado->Gerador->ExecProc();
codigo_objeto = ModuloDado->Gerador->ParamByName("Codigo_Retorno")->AsInteger;

if (ComboBox1_colonia->Text == "\0")
    tipo_col_instancias = NULL; // Objeto não instanciável, ou seja, ele já é
    // uma instância
else
    // Busca o código do tipo da colônia onde as instâncias do objeto irão habitar
    tipo_col_instancias = Buscar_TColonia_Por_Nome(ComboBox1_colonia->Text);

// Insere o código do objeto na base, associando todas as suas informações
Inserir_Objeto(codigo_objeto,tipo_objeto, ComboBox1_nome_objeto->Text,
    colonia_aberta, tipo_col_instancias);

// O comando if ... else a seguir trata os vínculos fixos a serem
// feitos no objeto caso ele tenha sido instanciado

// Código para caso Tipo_Obj == MetaTipoObjeto
if (tipo_corrente == 2)
    // Insere o nome do objeto na base
    Inserir_Valores_Texto(3,codigo_objeto,-1,ComboBox1_nome_objeto->Text);
    // Obs: A posição -1 indica objeto
else
{
    // Busca o código do atributo identificador do objeto pai a partir de seu código
    codigo_atributo = Buscar_Ident_Nivel_Especifico(tipo_corrente, 1);
    Inserir_Valores_Texto(codigo_atributo,codigo_objeto,-1,ComboBox1_nome_objeto->Text);
    Inserir_Vinculos_Fixos(codigo_objeto, tipo_corrente);
}

```

```

// Atualiza os grids de classificação e instanciação após inserção de objetos
ModuloDado->Grid_Classificacao->Close();
ModuloDado->Grid_Classificacao->ParamByName("Codigo")->AsInteger = codigo_objeto;
ModuloDado->Grid_Classificacao->Open();

// Atualiza os atributos de instanciação e seus valores no grid de instanciação
Seta_Valor_Grid();

if (MessageBox(0,"Objeto armazenado na base",
               "Confirmação", MB_OK)==IDOK)
    return;

} // Fim do botão: + Objeto

//-----
// Ação associada ao evento DropDown para o campo nome do objeto
void __fastcall TForm_Objeto::ComboBox1_nome_objetoDropDown(TObject *Sender)
{
    // Lista os nomes dos objetos do tipo corrente, que habitam uma colônia
    // aberta (cujo tipo é constrito pelo tipo de objeto corrente)
    Listar_Objeto(ComboBox1_nome_objeto,ComboBox_tipo_obj->Text);
}

//-----
// Funcionalidade: insere informações associadas a um objeto na tabela objeto
// Parâmetros de entrada: código do objeto, código do tipo do objeto, nome do
// objeto, código da colônia que habita, código do tipo da colônia das instâncias
void Inserir_Objeto(int codigo_objeto, int codigo_tipo, AnsiString nome_objeto,
                   int colonia_habita, int tipo_col_instancias)
{
    AnsiString parametro;

    TQuery *Inserir_objeto; // Declaração do componente TQuery local
    Inserir_objeto = new TQuery(Form_Objeto);

    Inserir_objeto->Active = false;
    Inserir_objeto->DatabaseName = "SIRIUS";

    parametro = "INSERT INTO objeto (CodObj, CodTObj, CodColHab, CodTCollInst)";
    parametro = parametro + " VALUES (:Obj, :Tipo, :Hab, :Inst)";
    Inserir_objeto->SQL->Clear();
    Inserir_objeto->SQL->Add(parametro);

    Inserir_objeto->ParamByName("Obj")->AsInteger = codigo_objeto;
    Inserir_objeto->ParamByName("Tipo")->AsInteger = codigo_tipo;
    Inserir_objeto->ParamByName("Hab")->AsInteger = colonia_habita;
    Inserir_objeto->ParamByName("Inst")->AsInteger = tipo_col_instancias;

    Inserir_objeto->ExecSQL();
    Inserir_objeto->Close();
    Inserir_objeto->UnPrepare();
    delete Inserir_objeto;
} // Fim da função: Inserir_Objeto

//-----
// Funcionalidade: insere o valor de um atributo com característica texto, ou o
// nome de um objeto, na tabela caract_texto
// Parâmetros de entrada: código do atributo, código do objeto, posição do valor,
// valor
void Inserir_Valor_Texto(int atributo,int objeto,int posicao,AnsiString valor)

```



```

{
    AnsiString parametro;

    TQuery *Inserir_Valor; // Declaração do componente TQuery local
    Inserir_Valor = new TQuery(Form_Objeto);

    Inserir_Valor->Active = false;
    Inserir_Valor->DatabaseName = "SIRIUS";

    parametro = "INSERT INTO caract_texto(CodTAt, CodObj, Posicao, Valor)";
    parametro = parametro + " VALUES (:Atrib, :Obj, :Pos, :Valor)";

    Inserir_Valor->SQL->Clear();
    Inserir_Valor->SQL->Add(parametro);
    Inserir_Valor->ParamByName("Atrib")->AsInteger = atributo;
    Inserir_Valor->ParamByName("Obj")->AsInteger = objeto;
    Inserir_Valor->ParamByName("Pos")->AsInteger = posicao;
    Inserir_Valor->ParamByName("Valor")->AsString = valor;

    Inserir_Valor->ExecSQL();
    Inserir_Valor->Close();
    Inserir_Valor->UnPrepare();
    delete Inserir_Valor;

} // Fim da função: Inserir_Valor_Texto

//-----
// Funcionalidade: insere o valor de um atributo com característica número na tabela
// caract_numero
// Parâmetros de entrada: código do atributo, código do objeto, posição do valor,
// valor
void Inserir_Valor_Numero(int objeto,int atributo,int posicao,AnsiString valor)
{
    AnsiString parametro;

    TQuery *Inserir_Valor; // Declaração do componente TQuery local
    Inserir_Valor = new TQuery(Form_Objeto);

    Inserir_Valor->Active = false;
    Inserir_Valor->DatabaseName = "SIRIUS";

    parametro = "INSERT INTO caract_numero(CodTAt, CodObj, Posicao, Valor)";
    parametro = parametro + " VALUES (:Atrib, :Obj, :Pos, :Valor)";

    Inserir_Valor->SQL->Clear();
    Inserir_Valor->SQL->Add(parametro);
    Inserir_Valor->ParamByName("Atrib")->AsInteger = atributo;
    Inserir_Valor->ParamByName("Obj")->AsInteger = objeto;
    Inserir_Valor->ParamByName("Pos")->AsInteger = posicao;
    Inserir_Valor->ParamByName("Valor")->AsInteger = StrToInt(valor);

    Inserir_Valor->ExecSQL();
    Inserir_Valor->Close();
    Inserir_Valor->UnPrepare();
    delete Inserir_Valor;

} // Fim da função: Inserir_Valor_Numero

//-----
// Funcionalidade insere o vínculo entre objeto e atributo na tabela
// vincula_atributo_objeto

```



```

// Parâmetros de entrada: código do atributo, código do objeto, nível de
// classificação/instanciação do atributo, identificador (S/N), limite inferior,
// limite superior, organização, definido (S/N), default (S/N), Fixo
void Inserir_Vinculo(int Atributo, int Objeto, int Nivel, AnsiString Ident,
                    int LI, int LS, AnsiString Org, AnsiString Definido,
                    AnsiString Default, int Fixo)
{
    AnsiString parametro;

    TQuery *Inserir_Vinculo; // Declaração do componente TQuery local
    Inserir_Vinculo = new TQuery(Form_Objeto);

    Inserir_Vinculo->Active = false;
    Inserir_Vinculo->DatabaseName = "SIRIUS";

    parametro = "INSERT INTO vincula_atributo_objeto";
    parametro = parametro + "(CodTat, CodObj, Nivel, Identificador, LI, LS, Organizacao, Definido, ValorDef, Fixo)";
    parametro = parametro + " VALUES (:Atrib, :Obj, :Niv, :Ident, :LI, :LS, :Org, :Def, :VDef, :Fix)";

    Inserir_Vinculo->SQL->Clear();
    Inserir_Vinculo->SQL->Add(parametro);
    Inserir_Vinculo->ParamByName("Atrib")->AsInteger = Atributo;
    Inserir_Vinculo->ParamByName("Obj")->AsInteger = Objeto;
    Inserir_Vinculo->ParamByName("Niv")->AsInteger = Nivel;
    Inserir_Vinculo->ParamByName("Ident")->AsString = Ident;
    Inserir_Vinculo->ParamByName("LI")->AsInteger = LI;
    Inserir_Vinculo->ParamByName("LS")->AsInteger = LS;
    Inserir_Vinculo->ParamByName("Org")->AsString = Org;
    Inserir_Vinculo->ParamByName("Def")->AsString = Definido;
    Inserir_Vinculo->ParamByName("VDef")->AsString = Default;
    Inserir_Vinculo->ParamByName("Fix")->AsInteger = Fixo;

    Inserir_Vinculo->ExecSQL();
    Inserir_Vinculo->Close();
    Inserir_Vinculo->UnPrepare();
    delete Inserir_Vinculo;
} // Fim da função: Inserir_Vinculo

//-----
// Funcionalidade: retorna código do objeto, buscando em colônia aberta
// Parâmetro de entrada: nome do objeto
// Retorno da função: código do objeto
int Buscar_Obj_Por_Nome(AnsiString nome)
{
    int codigo_objeto, codigo_colonia;
    AnsiString parametro;

    if (nome == "Meta Tipo Obj") // Para caso de buscas do código do Meta-Tipo
        return 2; // Retorna direto o seu código (2) sem verificar
                  // colônias

    // Componente para buscar o código do objeto
    TQuery *Buscar_Obj; // Declaração do componente TQuery local
    Buscar_Obj = new TQuery(Form_Objeto);

    Buscar_Obj->Active = false;
    Buscar_Obj->DatabaseName = "SIRIUS";

    // Componente para buscar as colônias abertas
    TQuery *Colonias_Abertas; // Declaração do componente TQuery local

```

```

Colonias_Abertas = new TQuery(Form_Objeto);

Colonias_Abertas->Active = false;
Colonias_Abertas->DatabaseName = "SIRIUS";

parametro = "SELECT CodCol FROM temporaria_colonia";

Colonias_Abertas->SQL->Clear();
Colonias_Abertas->SQL->Add(parametro);
Colonias_Abertas->Prepare();
Colonias_Abertas->ExecSQL();
Colonias_Abertas->Open();
Colonias_Abertas->First();

// Verifica se existe objeto com o nome especificado em alguma colônia aberta
while (!Colonias_Abertas->Eof)
{
    codigo_colonia = Colonias_Abertas->FieldByName("CodCol")->AsInteger;

    parametro = "";
    parametro = "SELECT o.CodObj FROM caract_texto t, objeto o";
    parametro = parametro + " WHERE t.Valor = :valor AND t.Posicao = -1"; // Posição de um objeto
    parametro = parametro + " AND t.CodObj = o.CodObj AND o.CodColHab = :Colonia";

    Buscar_Obj->SQL->Clear();
    Buscar_Obj->SQL->Add(parametro);
    Buscar_Obj->ParamByName("valor")->AsString = nome;
    Buscar_Obj->ParamByName("Colonia")->AsInteger = codigo_colonia;
    Buscar_Obj->Prepare();
    Buscar_Obj->ExecSQL();
    Buscar_Obj->Open();

    codigo_objeto = Buscar_Obj->FieldByName("CodObj")->AsInteger;
    // Se o nome do objeto já foi encontrado, direciona a busca para a
    // última colônia aberta
    if (codigo_objeto != 0)
        Colonias_Abertas->Last();

    Colonias_Abertas->Next();
} // Fim do comando while

Buscar_Obj->Close();
Buscar_Obj->UnPrepare();
delete Buscar_Obj;

Colonias_Abertas->Close();
Colonias_Abertas->UnPrepare();
delete Colonias_Abertas;

return codigo_objeto;

} // Fim da função: Buscar_Obj_Por_Nome

//-----
// Funcionalidade: retorna o nome do objeto a partir de seu código
// Parâmetro de entrada: código do objeto
// Retorno da função: nome do objeto
AnsiString Buscar_Obj_Por_Codigo (int codigo)
{
    AnsiString nome, parametro;

```

```
TQuery *Buscar_Obj; // Declaração do componente TQuery local
Buscar_Obj = new TQuery(Form_Objeto);

Buscar_Obj->Active = false;
Buscar_Obj->DatabaseName = "SIRIUS";

parametro = "SELECT Valor FROM caract_texto, objeto";
parametro = parametro + " WHERE objeto.CodObj = :Codigo_Obj";
parametro = parametro + " AND caract_texto.CodObj = objeto.CodObj";

Buscar_Obj->SQL->Clear();
Buscar_Obj->SQL->Add(parametro);

Buscar_Obj->ParamByName("Codigo_Obj")->AsInteger = codigo;
Buscar_Obj->Prepare();
Buscar_Obj->ExecSQL();
Buscar_Obj->Open();
nome = Buscar_Obj->FieldByName("Valor")->AsString;
Buscar_Obj->Close();
Buscar_Obj->UnPrepare();
delete Buscar_Obj;
return nome;

} // Fim da função: Buscar_Obj_Por_Codigo

//-----
// Ação associada ao evento click (escolha de objeto) para campo nome do objeto
void __fastcall TForm_Objeto::ComboBox1_nome_objetoClick(TObject *Sender)
{
    int codigo_objeto;
    AnsiString parametro, nome_colonia;

    // Verifica se existe item selecionado
    if (ComboBox1_nome_objeto->ItemIndex != -1)
    {
        TQuery *Buscar_Obj; // Declaração do componente TQuery local
        Buscar_Obj = new TQuery(Form_Objeto);

        Buscar_Obj->Active = false;
        Buscar_Obj->DatabaseName = "SIRIUS";

        codigo_objeto = Buscar_Obj_Por_Nome(Form_Objeto->ComboBox1_nome_objeto->Text);

        parametro = "SELECT NomeCol FROM objeto o, tipo_colonia c";
        parametro = parametro + " WHERE o.CodObj = :Codigo_Obj";
        parametro = parametro + " AND o.CodTCollnst = c.CodTCol";

        Buscar_Obj->SQL->Clear();
        Buscar_Obj->SQL->Add(parametro);
        Buscar_Obj->ParamByName("Codigo_Obj")->AsInteger = codigo_objeto;

        Buscar_Obj->Prepare();
        Buscar_Obj->ExecSQL();
        Buscar_Obj->Open();
        nome_colonia = Buscar_Obj->FieldByName("NomeCol")->AsString;
        Buscar_Obj->Close();
        Buscar_Obj->UnPrepare();
        delete Buscar_Obj;

        // Lista os tipos de colônias no campo tipo de colônia c
        // seleciona o nome encontrado acima (nome_colonia)
```



```

    Listar_Tipos_Colonias(ComboBox1_colonia);
    for (int i=0;i<=0;i++)
        if (ComboBox1_colonia->Items->Strings[i] == nome_colonia)
        {
            ComboBox1_colonia->ItemIndex = i;
            i = -2; // Quebra o loop do for quando a colônia procurada é encontrada
        }

    // Mostra atributos de Classificacao
    ModuloDado->Grid_Classificacao->Close();
    ModuloDado->Grid_Classificacao->ParamByName("Codigo")->AsInteger = codigo_objeto;
    ModuloDado->Grid_Classificacao->Open();
    // Atualiza os atributos de instanciação no grid de instanciação
    Seta_Valor_Grid();
} // Fim do if item selecionado

} // Fim do evento click no campo nome de objeto

//-----
// Funcionalidade: busca o código do atributo identificador do objeto
// Parâmetros de entrada: código do objeto, nível de classificação/instanciação
// Retorno da função: código do atributo identificador
int Buscar_Ident_Nivel_Especifico(int cod_obj, int nivel)
{
    int codigo_atributo;
    AnsiString parametro;

    TQuery *Buscar_Obj; // Declaração do componente TQuery local
    Buscar_Obj = new TQuery(Form_Objeto);

    Buscar_Obj->Active = false;
    Buscar_Obj->DatabaseName = "SIRIUS";

    parametro = "SELECT CodTat FROM vincula_atributo_objeto";
    parametro = parametro + " WHERE CodObj = :Codigo_Obj";
    parametro = parametro + " AND Identificador = :Ident";
    parametro = parametro + " AND Nivel = :niv";

    Buscar_Obj->SQL->Clear();
    Buscar_Obj->SQL->Add(parametro);

    Buscar_Obj->ParamByName("Codigo_Obj")->AsInteger = cod_obj;
    Buscar_Obj->ParamByName("Ident")->AsString = "S";
    Buscar_Obj->ParamByName("niv")->AsInteger = nivel;
    Buscar_Obj->Prepare();
    Buscar_Obj->ExecSQL();

    Buscar_Obj->Open();
    codigo_atributo = Buscar_Obj->FieldByName("CodTat")->AsInteger;

    Buscar_Obj->Close();
    Buscar_Obj->UnPrepare();
    delete Buscar_Obj;

    return codigo_atributo;

} // Fim da função: Buscar_Atrib_Identificador

//-----
// Funcionalidade: lista os objetos de um determinado tipo num componente
// Parâmetros de entrada: componente (onde são listados os objetos) e nome do

```

```
// tipo de objeto
void Listar_Objeto (TComboBox *Componente, AnsiString nome_tipo)
{
    AnsiString parametro;
    int codigo_tipo, colonia_aberta, tipo_col_habita;

    TQuery *Listar_Objeto; // Declaração do componente TQuery local
    Listar_Objeto = new TQuery(Form_Objeto);

    Listar_Objeto->Active = false;
    Listar_Objeto->DatabaseName = "SIRIUS";

    Componente->Clear();

    // Busca o código do tipo de objeto corrente
    codigo_tipo = Buscar_Obj_Por_Nome(nome_tipo);

    // Busca o tipo da colônia onde as instâncias do tipo de objeto corrente habitam
    tipo_col_habita = Buscar_TColonia_Instanceias(codigo_tipo);

    // Recupera, se houver, o código da colônia aberta do tipo de colônia encontrado
    colonia_aberta = Colonia_Aberta(tipo_col_habita);

    parametro = "SELECT Valor FROM caract_texto t, objeto o";
    parametro = parametro + " WHERE t.CodObj = o.CodObj AND o.CodObj > 2";
    parametro = parametro + " AND t.posicao = -1 AND o.CodTObj = :Tipo";
    parametro = parametro + " AND o.CodColHab = :Colonia";

    Listar_Objeto->SQL->Clear();
    Listar_Objeto->SQL->Add(parametro);

    Listar_Objeto->ParamByName("Tipo")->AsInteger = codigo_tipo;
    Listar_Objeto->ParamByName("Colonia")->AsInteger = colonia_aberta;
    Listar_Objeto->Prepare();
    Listar_Objeto->ExecSQL();

    // Adiciona os objetos encontrados no componente
    Listar_Objeto->Open();
    Listar_Objeto->First();
    while (!Listar_Objeto->Eof)
    {
        AnsiString item;
        item = Listar_Objeto->FieldByName("Valor")->AsString;
        Componente->Items->Add(item);
        Listar_Objeto->Next();
    }

    Listar_Objeto->Close();
    Listar_Objeto->UnPrepare();
    delete Listar_Objeto;
} // Fim da função: Listar_Objeto

//-----
// Nome do botão: Instanciar
// Funcionalidade: verifica se o objeto corrente pode ser instanciado
void __fastcall TForm_Objeto::Button_instanciarClick(TObject *Sender)
{
    int codigo_objeto;

    // Testa se o campo nome do objeto esta preenchido
```

```

if (Form_Objeto->ComboBox1_nome_objeto->Text == "\0")
{
    if (MessageBox(0,"O objeto a ser instanciado não foi escolhido",
        "Informação incompleta", MB_OK)==IDOK)
        return;
}
codigo_objeto = Buscar_Obj_Por_Nome(ComboBox1_nome_objeto->Text);

// Verifica se o objeto escolhido pode ser instanciado
// Verifica se ele possui atributo identificador
if (Buscar_Ident_Nivel_Especifico(codigo_objeto, 1) == 0)
    if (MessageBox(0,"O objeto escolhido não possui identificador",
        "O objeto especificado não pode ser instanciado", MB_OK)==IDOK)
        return;

// Verifica se o objeto escolhido pode ser instanciado
// Verifica se ele possui tipo de colônia para as instâncias
if (ComboBox1_colonia->Text == "\0")
    if (MessageBox(0,"O objeto escolhido não possui tipo de colônia definido para suas instâncias",
        "O objeto especificado não pode ser instanciado", MB_OK)==IDOK)
        return;
Listar_Tipos_Obj(ComboBox_tipo_obj);

// Verifica se o objeto a ser instanciado já é tipo. Se não for, adiciona o
// nome do objeto na lista de tipos, no campo de tipo de objeto
if (!Verifica_Tipo_Objeto(codigo_objeto))
    ComboBox_tipo_obj->Items->Add(ComboBox1_nome_objeto->Text);

// Seleciona o nome do objeto a ser instanciado no campo de tipo de objeto,
// ou seja, transfere o nome do objeto para o tipo do objeto
for (int i=0; i<=0; i++)
    if (ComboBox_tipo_obj->Items->Strings[i] == ComboBox1_nome_objeto->Text)
    {
        ComboBox_tipo_obj->ItemIndex = i;
        i = -2;    // Quebra o loop do for
    }

// Limpa campo colônia
ComboBox1_colonia->ItemIndex = -1;
// Limpa o campo do nome do objeto para que outro objeto seja editado
ComboBox1_nome_objeto->Text = "\0";
ModuloDado->Grid_Classificacao->Close();
Limpa_Grid_Instanceia();

} // Fim do botão Instanciar

//-----
// Funcionalidade: limpa a tela de objeto
void Limpa_Tela_Objeto(void)
{
    Form_Objeto->ComboBox1_nome_objeto->Text = "\0";
    // Seleciona o Meta Tipo Obj como tipo de objeto
    Listar_Tipos_Obj(Form_Objeto->ComboBox_tipo_obj);
    for (int i=0; i<=0; i++)
        if (Form_Objeto->ComboBox_tipo_obj->Items->Strings[i] == "Meta Tipo Obj")
        {
            Form_Objeto->ComboBox_tipo_obj->ItemIndex = i;
            i = -2;
        }
    Form_Objeto->ComboBox1_colonia->Text = "\0";
    ModuloDado->Grid_Classificacao->Close();
}

```



```

// Dá nome às colunas do grid de instanciação
Form_Objeto->Grid_Instanc->Cells[0][0] = "ATRIBUTO";
Form_Objeto->Grid_Instanc->Cells[1][0] = "SUB_ATRIBUTO";
Form_Objeto->Grid_Instanc->Cells[2][0] = "CARACT.";
Form_Objeto->Grid_Instanc->Cells[3][0] = "VALOR";

Limpa_Grid_Instancia();

} // Fim da função: Limpa_Tela_Objeto

//-----
// Nome do Botão: Editar Valor (At. Classif)
void __fastcall TForm_Objeto::Button_edita_valor_classifClick(TObject *Sender)
{
    // Limpa os campos da tela edita valor classificação
    Limpa_Editar_Valores_Classif();
    // Chama o formulário Editar_Valores_Classif (Edita Valor de Atributo (Objeto
    // não instanciado))
    Form_Editar_Valores_Classif->ShowModal();
}
//-----
// Nome do Botão: Editar Valor (At. Instanc.)
void __fastcall TForm_Objeto::Button_edita_valor_instClick(TObject *Sender)
{
    // Limpa os campos da tela edita valor instancia
    Limpa_Editar_Valores();
    // Chama o formulário Editar_Valores (Edita Valor de Atributo (Objeto instanciado))
    Form_Editar_Valores->ShowModal();
}
//-----
// Nome do Botão: Def. Valor
void __fastcall TForm_Objeto::Button_define_valorClick(TObject *Sender)
{
    // Limpa os campos da tela define valor
    Limpa_Def_Valores();
    // Chama o formulário Def_Valores (Edita Valor de Atributo (Atributo definido
    // no pai))
    Form_Def_Valores->ShowModal();
}
//-----
// Nome do Botão: Def. Atr. Extra
void __fastcall TForm_Objeto::Button_define_atrib_extraClick(TObject *Sender)
{
    // Atualiza o grid de atributos
    ModuloDado->Grid_Atributos->Close();
    ModuloDado->Grid_Atributos->Open();

    // Chama o formulário Opcao (Define/Vincula Atributo)
    Form_Opcao->ShowModal();
    Form_Opcao->Button_vincula->Enabled = true;
}
//-----
// Funcionalidade: insere os vinculos fixos do objeto pai ao objeto filho
// Parâmetros de entrada: código do objeto filho, código o objeto pai
void Inserir_Vinculos_Fixos(int codigo_filho, int codigo_pai)
{
    int codigo_atributo, nivel, inf, sup, fixo;
    AnsiString org, ident, definido, valor_def, valor, parametro, caracteristica;

```

```
struct Multivalorado ultima_posicao;

TQuery *Busca_Fixos; // Declaração do componente TQuery local
Busca_Fixos = new TQuery(Form_Objeto);

Busca_Fixos->Active = false;
Busca_Fixos->DatabaseName = "SIRIUS";

// Selecionar todos os atributos fixos e de nível >= 2 do pai (cod_obj),
// com exceção do atributo identificador de nível 1

parametro = "SELECT CodTat, Nivel, Identificador, LI, LS, Organizacao, Definido, ValorDef";
parametro = parametro + " FROM vincula_atributo_objeto";
parametro = parametro + " WHERE CodObj = :Codigo_Obj";
parametro = parametro + " AND Nivel >= 2";
parametro = parametro + " UNION SELECT CodTat, Nivel, Identificador, LI, LS, Organizacao, Definido, ValorDef";
parametro = parametro + " FROM vincula_atributo_objeto";
parametro = parametro + " WHERE CodObj = :Codigo_Obj";
parametro = parametro + " AND Identificador = :NaoIdent AND Nivel = 1 AND Fixo != 0";

Busca_Fixos->SQL->Clear();
Busca_Fixos->SQL->Add(parametro);

Busca_Fixos->ParamByName("Codigo_Obj")->AsInteger = codigo_pai;
Busca_Fixos->ParamByName("NaoIdent")->AsString = "N";
Busca_Fixos->Prepare();
Busca_Fixos->ExecSQL();

Busca_Fixos->Open();
Busca_Fixos->First();

// Se o atributo é fixo no pai, então também é fixo no filho, ou seja,
// a informação "fixo" no filho tem como valor o código do filho
fixo = codigo_filho;

// Busca as informações dos atributos fixos do pai e vincula ao filho
while (!Busca_Fixos->Eof)
{
    codigo_atributo = Busca_Fixos->FieldByName("CodTat")->AsInteger;
    ident = Busca_Fixos->FieldByName("Identificador")->AsString;
    valor_def = Busca_Fixos->FieldByName("ValorDef")->AsString;
    nivel = Busca_Fixos->FieldByName("Nivel")->AsInteger;
    if (nivel > 0)
        nivel--; // Decrementa o nível do atributo no filho
    inf = Busca_Fixos->FieldByName("LI")->AsInteger;
    sup = Busca_Fixos->FieldByName("LS")->AsInteger;
    org = Busca_Fixos->FieldByName("Organizacao")->AsString;
    definido = Busca_Fixos->FieldByName("Definido")->AsString;

    // Insere o vínculo de cada atributo fixo ao objeto filho
    Inserir_Vinculo(codigo_atributo, codigo_filho, nivel, ident, inf, sup,
        org, definido, valor_def, fixo);

    // Se o atributo for default, passa para o filho o valor, se existir
    if (valor_def == "S")
    {
        // Busca a característica do atributo
        caracteristica = Buscar_Caract_Atrib(codigo_atributo);
        // Busca a última posição de valor do atributo
        // Monovalorado: última posição = 0
        // Multivalorado: posição do último valor inserido
    }
}
```

```

ultima_posicao = Buscar_Ultima_Posicao(codigo_pai, codigo_atributo);
for(int i=0; i<=ultima_posicao.posicao; i++)
{
    if ((valor = Buscar_Valor(codigo_pai, codigo_atributo, i, caracteristica))!= "\0")
        if (caracteristica == "Texto")
            // Insere um valor de caracteristica texto
            Inserir_Valor_Texto(codigo_atributo, codigo_filho, i, valor);

            else if (caracteristica == "Número")
                // Insere um valor de caracteristica número
                Inserir_Valor_Numero(codigo_filho, codigo_atributo, i, valor);
    } // Fim do for
} // Fim do if default

Busca_Fixos->Next();
} // Fim while

Busca_Fixos->Close();
Busca_Fixos->UnPrepare();
delete Busca_Fixos;

} // Fim da função: Inserir_Vinculos_Fixos

//-----
// Funcionalidade: verifica se o objeto a ser removido é tipo, ou seja,
// se possui instâncias
// Parâmetros de entrada: código do objeto
// Retorno: true se o objeto é tipo ou false se o objeto não é tipo
bool Verifica_Tipo_Objeto(int Codigo)
{
    AnsiString parametro;

    TQuery *Buscar_Obj; // Declaração do componente TQuery local
    Buscar_Obj = new TQuery(Form_Objeto);

    Buscar_Obj->Active = false;
    Buscar_Obj->DatabaseName = "SIRIUS";

    parametro = "SELECT CodTObj FROM objeto";
    parametro = parametro + " WHERE CodTObj = :Cod";

    Buscar_Obj->SQL->Clear();
    Buscar_Obj->SQL->Add(parametro);

    Buscar_Obj->ParamByName("Cod")->AsInteger = Codigo;
    Buscar_Obj->Prepare();
    Buscar_Obj->ExecSQL();

    Buscar_Obj->Open();

    if (( Buscar_Obj->FieldByName("CodTObj")->AsInteger)!= 0) // É Tipo
    {
        Buscar_Obj->Close();
        Buscar_Obj->UnPrepare();
        delete Buscar_Obj;

        return true;
    }

    Buscar_Obj->Close();
    Buscar_Obj->UnPrepare();
    delete Buscar_Obj;
}

```



```
return false; // O objeto não é tipo

} // Fim da função: Verifica_Tipo_Objeto

//-----
// Nome do Botão: Classificar
// Funcionalidade: classifica o objeto corrente
void __fastcall TForm_Objeto::Button_classificarClick(TObject *Sender)
{
    int codigo_objeto, codigo_tipo, tipo_colonia;
    AnsiString nome_tipo_colonia;

    // Verifica se o objeto corrente pode ser classificado
    if (ComboBox_tipo_obj->Text == "Meta Tipo Obj")
        if (MessageBox(0, "O objeto está no nível mais alto na hierarquia de classificação",
            "O objeto não pode ser classificado", MB_OK)==IDOK)
            return;

    // Transfere o tipo para o campo nome do objeto corrente
    ComboBox1_nome_objeto->Text = ComboBox_tipo_obj->Text;

    // Busca o código e o tipo do objeto classificado
    codigo_objeto = Buscar_Obj_Por_Nome(ComboBox1_nome_objeto->Text);
    codigo_tipo = Buscar_Tipo_Obj(codigo_objeto);

    // Seleciona o nome do tipo no campo do tipo objeto
    Listar_Tipos_Obj(ComboBox_tipo_obj);
    for (int i=0; i<ComboBox_tipo_obj->Items->Count; i++)
        if (ComboBox_tipo_obj->Items->Strings[i] == (Buscar_Obj_Por_Codigo(codigo_tipo)))
        {
            ComboBox_tipo_obj->ItemIndex = i;
            i = -2; // Quebra o loop do for
        }

    // Busca o código e o nome do tipo da colônia onde as instâncias do objeto
    // classificado habitam
    tipo_colonia = Buscar_TColonia_Instanceias(codigo_objeto);
    nome_tipo_colonia = Buscar_TColonia_Por_Codigo(tipo_colonia);

    // Seleciona o nome do tipo da colônia no campo tipo colônia
    Listar_Tipos_Colonias(ComboBox1_colonia);
    for (int i=0; i<ComboBox1_colonia->Items->Count; i++)
        if (ComboBox1_colonia->Items->Strings[i] == nome_tipo_colonia)
        {
            ComboBox1_colonia->ItemIndex = i;
            i = -2; // Quebra o loop do for
        }

    //Atualiza o grid classificação após serem feitos os vínculos necessários
    ModuloDado->Grid_Classificacao->Close();
    ModuloDado->Grid_Classificacao->ParamByName("Codigo")->AsInteger = codigo_objeto;
    ModuloDado->Grid_Classificacao->Open();

    // Atualiza os atributos de instanciação no grid de instanciação
    Seta_Valores_Grid();
} // Fim do evento: Button_classificarClick

//-----
// Funcionalidade: busca o tipo de um objeto
```

```
// Parâmetro de entrada: código do objeto
// Retorno: código do tipo do objeto
int Buscar_Tipo_Obj(int objeto)
{
    AnsiString parametro;
    int tipo;

    TQuery *Buscar_Tipo; // Declaração do componente TQuery local
    Buscar_Tipo = new TQuery(Form_Objeto);

    Buscar_Tipo->Active = false;
    Buscar_Tipo->DatabaseName = "SIRIUS";

    parametro = "SELECT CodTObj FROM objeto";
    parametro = parametro + " WHERE CodObj = :Cod_Obj";

    Buscar_Tipo->SQL->Clear();
    Buscar_Tipo->SQL->Add(parametro);

    Buscar_Tipo->ParamByName("Cod_Obj")->AsInteger = objeto;
    Buscar_Tipo->Prepare();
    Buscar_Tipo->ExecSQL();

    Buscar_Tipo->Open();
    tipo = Buscar_Tipo->FieldByName("CodTObj")->AsInteger;

    Buscar_Tipo->Close();
    Buscar_Tipo->UnPrepare();
    delete Buscar_Tipo;

    return tipo;
} // Fim da função: Buscar_Tipo_Obj

//-----
// Funcionalidade: atualiza valores de instânciação no grid
void Seta_Valor_Grid(void)
{
    AnsiString parametro, característica, valor;
    int objeto, atributo, j, sub_atributo;
    struct Multivalorado ultima_posicao;

    TQuery *Atrib_Instancia; // Declaração do componente TQuery local
    Atrib_Instancia = new TQuery(Form_Objeto);

    TQuery *Sub_atributos; // Declaração do componente TQuery local
    Sub_atributos = new TQuery(Form_Objeto);

    Sub_atributos->Active = false;
    Sub_atributos->DatabaseName = "SIRIUS";

    Atrib_Instancia->Active = false;
    Atrib_Instancia->DatabaseName = "SIRIUS";

    // Limpa o grid de instânciação
    Limpa_Grid_Instancia();

    j=1; // Variável que indica a linha do grid de instânciação
    objeto = Buscar_Obj_Por_Nome(Form_Objeto->ComboBox1_nome_objeto->Text);

    // Seleciona os atributos de instânciação
```

```

parametro = "SELECT CodTAt, NomeAt, Caracteristica";
parametro = parametro + " FROM tipo_atributo a, vincula_atributo_objeto v";
parametro = parametro + " WHERE v.CodObj = :Obj AND v.Nivel = 0 AND v.CodTAt != 3";
parametro = parametro + " AND a.CodTAt = v.CodTAt";
parametro = parametro + " ORDER BY NomeAt";

Atrib_Instancia->SQL->Clear();
Atrib_Instancia->SQL->Add(parametro);
Atrib_Instancia->ParamByName("Obj")->AsInteger = objeto;
Atrib_Instancia->Prepare();
Atrib_Instancia->ExecSQL();

Atrib_Instancia->Open();
Atrib_Instancia->First();
// Para cada atributo, insere no grid de instanciação informações de nome do
// atributo, característica, sub atributos (se houver) e valores
while (!Atrib_Instancia->Eof)
{
    atributo = Atrib_Instancia->FieldByName("CodTAt")->AsInteger;
    // Busca a característica do atributo
    caracteristica = Atrib_Instancia->FieldByName("Caracteristica")->AsString;
    if (caracteristica != "Tupla" && caracteristica != "Atributo de atributo")
    { // O atributo não possui sub atributos
        // Busca a última posição alocada para um valor do atributo
        ultima_posicao = Buscar_Ultima_Posicao(objeto, atributo);
        // Para cada posição, insere no grid o valor correspondente
        for(int i=0; i<=ultima_posicao.posicao; i++)
        {
            valor = Buscar_Valor(objeto, atributo, i, caracteristica);
            if (valor != "\0" || ultima_posicao.posicao == 0)
            // if (valor != "\0")
            { // Insere as informações no Grid
                Form_Objeto->Grid_Instanc->Cells[0][j] = Atrib_Instancia->FieldByName("NomeAt")->AsString;
                Form_Objeto->Grid_Instanc->Cells[2][j] = caracteristica;
                Form_Objeto->Grid_Instanc->Cells[3][j] = valor;
                j++;
                // Incrementa o número de linhas do grid, se necessário
                if(j >= Form_Objeto->Grid_Instanc->RowCount)
                    Form_Objeto->Grid_Instanc->RowCount = (Form_Objeto->Grid_Instanc->RowCount) + 1;
            } // Fim do if valor != "\0", significa que existe um valor para se mostrado
        } // Fim do for
    } // Fim do if caracteristica (texto ou número)
    else // Característica tupla ou atributo de atributo
    {
        // Coloca no grid o atributo do tipo tupla ou atributo de atributo
        Form_Objeto->Grid_Instanc->Cells[0][j] = Atrib_Instancia->FieldByName("NomeAt")->AsString;
        Form_Objeto->Grid_Instanc->Cells[2][j] = caracteristica;
        j++;
        // Incrementa o número de linhas do grid, se necessário
        if(j >= Form_Objeto->Grid_Instanc->RowCount)
            Form_Objeto->Grid_Instanc->RowCount = (Form_Objeto->Grid_Instanc->RowCount) + 1;

        // Busca os sub atributos do atributo
        Sub_atributos = Buscar_Sub_Atributos(atributo);
        Sub_atributos->Open();
        Sub_atributos->First();
        // Para cada sub atributo, insere no grid de instanciação
        // as informações correspondentes
        while (!Sub_atributos->Eof)
        {
            sub_atributo = Sub_atributos->FieldByName("CodTAt")->AsInteger;

```



```

        caractéristica = Sub_atributos->FieldByName("Característica")->AsString;
        // Posição dos sub atributos é zero, estamos considerando apenas sub atributos
        // monovalorados
        // Busca o valor do sub atributo
        valor = Buscar_Valor(objeto, sub_atributo, 0, caractéristica);
        // Insere as informações as informações no grid
        Form_Objeto->Grid_Instanc->Cells[0][j] = Atrib_Instancia->FieldByName("NomeAt")->AsString;
        Form_Objeto->Grid_Instanc->Cells[1][j] = Sub_atributos->FieldByName("NomeAt")->AsString;
        Form_Objeto->Grid_Instanc->Cells[2][j] = caractéristica;
        Form_Objeto->Grid_Instanc->Cells[3][j] = valor;
        j++;
        // Incrementa o número de linhas do grid, se necessário
        if(j >= Form_Objeto->Grid_Instanc->RowCount)
            Form_Objeto->Grid_Instanc->RowCount = (Form_Objeto->Grid_Instanc->RowCount) + 1;
        Sub_atributos->Next(); // Busca o próximo sub atributo
    } // Fim do while que recupera todos os sub atributos
} // Fim do else (sub atributos)

Atrib_Instancia->Next(); // Busca o próximo atributo

} // Fim do while que recupera todos os nomes dos atributos instanciados

Atrib_Instancia->Close();
Atrib_Instancia->UnPrepare();
Sub_atributos->Close();
delete Sub_atributos;
delete Atrib_Instancia;

} // Fim da função: Seta_Valor_Grid

//-----
// Funcionalidade: lista o(s) sub atributo(s) de um atributo com característica
// tupla ou atributo de atributo
// Parâmetro de entrada: código do atributo
// Retorno: Componente TQuery contendo as informações do(s) sub atributo(s)
TQuery *Buscar_Sub_Atributos(int atributo)
{
    int obj_prim;
    AnsiString nome_atrib, parametro;

    TQuery *Busca_Sub_Atributos; // Declaração do componente TQuery local
    Busca_Sub_Atributos = new TQuery(Form_Objeto);

    Busca_Sub_Atributos->Active = false;
    Busca_Sub_Atributos->DatabaseName = "SIRIUS";

    // Busca o código do objeto primário que vincula o atributo a seu(s) sub atributo(s)
    obj_prim = Buscar_ObjPrimario(atributo);

    parametro = "SELECT CodTAt, NomeAt, Característica FROM tipo_atributo a, vincula_atrib_obj_primario v";
    parametro = parametro + " WHERE CodObjP = :ObjPrim AND v.CodTAt != :Atrib";
    parametro = parametro + " AND a.CodTAt = v.CodTAt";

    Busca_Sub_Atributos->SQL->Clear();
    Busca_Sub_Atributos->SQL->Add(parametro);
    Busca_Sub_Atributos->ParamByName("ObjPrim")->AsInteger = obj_prim;
    Busca_Sub_Atributos->ParamByName("Atrib")->AsInteger = atributo;
    Busca_Sub_Atributos->Prepare();
    Busca_Sub_Atributos->ExecSQL();

    return Busca_Sub_Atributos; // Retorna o endereço do componente TQuery
}

```

```

} // Fim da função: Listar_Sub_Atributos

//-----
// Funcionalidade: limpa o grid de instanciação
void Limpa_Grid_Instancia(void)
{
    // Número máximo de linhas do grid de instanciação
    int max_linha = Form_Objeto->Grid_Instanc->RowCount;

    for (int k=0; k<=3; k++) // Coluna
        for (int i=1; i<=max_linha; i++) // Linha
            Form_Objeto->Grid_Instanc->Cells[k][i] = "\0";

    Form_Objeto->Grid_Instanc->RowCount = 6;
} // Fim da função: Limpa_Grid_Instancia

//-----
// Nome do Botão: Fechar
// Funcionalidade: fecha as colônias abertas e sai do formulário objeto,
// retornando para o menu principal
void __fastcall TForm_Objeto::BitBtn_fecharClick(TObject *Sender)
{
    // Fecha as colônias abertas
    Fechar_Colonias();
    Form_Colonias_Abertas->Close();
}

//-----
// Ação associada ao evento DropDown para o campo tipo de colônia
void __fastcall TForm_Objeto::ComboBox1_coloniaDropDown(TObject *Sender)
{
    // Lista os tipos de colônias
    Listar_Tipos_Colonias(Form_Objeto->ComboBox1_colonia);
}

//-----
// Nome do Botão: Topo
// Funcionalidade: Retorna para o nível mais alto na hierarquia de classificação
void __fastcall TForm_Objeto::BitBtn_TopoClick(TObject *Sender)
{
    // Seleciona o Meta Tipo Objeto no campo de tipo objeto
    Listar_Tipos_Obj(ComboBox_tipo_obj);
    for (int i=0; i<=0; i++)
        if (ComboBox_tipo_obj->Items->Strings[i] == "Meta Tipo Obj")
        {
            ComboBox_tipo_obj->ItemIndex = i;
            i = -2; // Quebra o loop do for
        }

    ComboBox1_nome_objeto->Text = "\0";
    ComboBox1_colonia->Clear();
    // Limpa o Grid_Classificação
    ModuloDado->Grid_Classificacao->Close();
    Limpa_Grid_Instancia();
} // Fim do botão: Topo

//-----
// Nome do Botão: Define Colônia
// Funcionalidade: chama o formulário Tipo_Colonia (Define Tipo Colônia)
// para definição de tipo de colônia

```

```

void __fastcall TForm_Objeto::BitBtn_def_coloniaClick(TObject *Sender)
{
    Form_Tipo_Colonia->ComboBox_obj_constringe->ItemIndex = -1;
    Form_Tipo_Colonia->ComboBox_tipo_colonia->Text = "\0";
    Form_Tipo_Colonia->ShowModal();
} // Fim do botão: Define Colônia

//-----
// Nome do Botão: Habitar
// Funcionalidade: chama o formulário Habitar (Habita), para mostrar a colônia
// onde cada objeto da base habita
void __fastcall TForm_Objeto::Button_habitarClick(TObject *Sender)
{
    ModuloDado->Grid_Habitar->Close();
    ModuloDado->Grid_Habitar->Open();
    Form_Habitar->ShowModal();
} // Fim do botão: Habitar

//-----
// Nome do Botão: Generalizar
// Funcionalidade: chama o formulário Generalização (Generalização), para mostrar
// informações da hierarquia de generalização, de acordo com o tipo do objeto
// corrente
void __fastcall TForm_Objeto::Button_generalizarClick(TObject *Sender)
{
    // Transfere informações do objeto para o objeto genérico no formulário
    // generalização
    Form_Generalizacao->Edit_tipo_generico->Text = ComboBox_tipo_obj->Text;
    Form_Generalizacao->Edit_colonia_generico->Text = ComboBox1_colonia->Text;
    Form_Generalizacao->ComboBox_obj_generico->ItemIndex = -1;

    // Limpa restrições no formulário generalização
    Form_Generalizacao->RadioGroup1_sobreposicao->ItemIndex = -1;
    Form_Generalizacao->RadioGroup1_participacao->ItemIndex = -1;

    // Limpa os campos de critério e opção de generalização
    Form_Generalizacao->ComboBox_critério->Clear();
    Form_Generalizacao->ComboBox_opcao->Clear();

    // Limpa o grid de objetos específicos
    ModuloDado->Grid_Generalizar->Close();

    // Chama o formulário generalização
    Form_Generalizacao->ShowModal();
} // Fim do botão: Generalizar

//-----
// Ação associada ao evento DropDown para o campo tipo de objeto
void __fastcall TForm_Objeto::ComboBox_tipo_objDropDown(TObject *Sender)
{
    // Lista os tipos de objetos no campo tipo objeto
    Listar_Tipos_Obj(ComboBox_tipo_obj);
    ComboBox1_nome_objeto->Text = "\0";
}
//-----
// Funcionalidade: lista os tipos de objetos
// Parâmetro de entrada: componente onde são listados os tipos de objetos
void Listar_Tipos_Obj(TComboBox *Componente)
{
    AnsiString parametro, item;

```



```
TQuery *Listar_Tipos; // Declaração do componente TQuery local
Listar_Tipos = new TQuery(Form_Objeto);

Listar_Tipos->Active = false;
Listar_Tipos->DatabaseName = "SIRIUS";

Componente->Clear();

// Seleciona todos os objetos que são tipo
parametro = "SELECT DISTINCT Valor FROM caract_texto t, objeto o";
parametro = parametro + " WHERE t.CodObj = o.CodTObj AND t.Posicao = -1";

Listar_Tipos->SQL->Clear();
Listar_Tipos->SQL->Add(parametro);
Listar_Tipos->Prepare();
Listar_Tipos->ExecSQL();

Listar_Tipos->Open();
Listar_Tipos->First();
// Para cada tipo, insere seu nome no componente
while (!Listar_Tipos->Eof)
{
    item = Listar_Tipos->FieldByName("Valor")->AsString;
    if (item != "\0")
        Componente->Items->Add(item);
    Listar_Tipos->Next();
}
if (Componente->Items->Strings[0] == "\0")
    Componente->Items->Add("Meta Tipo Obj");
Listar_Tipos->Close();
Listar_Tipos->UnPrepare();
delete Listar_Tipos;

} // Fim da função: Listar_Tipos_Obj
//_____
```

Referências Bibliográficas

- [Araujo_1999] Araujo, M. R. B; Traina, C. Jr - "Editor de Esquemas - ESIRIUS para o Modelo SIRIUS Manual do Usuário", Relatório Técnico do ICMC - USP, nro 88, Abril de 1999.
- [Araujo_1998] Araujo, M. R B. - "Representação de Construtores Semânticos em SIIRIUS e Suporte através de um Editor de Esquemas", Dissertação de Mestrado em Ciências da Computação ICMC - USP, São Carlos - SP, Agosto de 1998.
- [Araujo_1998a] Araujo, M. R. B; Traina, C. Jr - "Editor de Esquemas com Suporte para Hierarquia de Classificação", XIII Simpósio Brasileiro de Banco de Dados, realizado em Maringá, de 12 a 14 de outubro de 1998.
- [Biajiz_1996] Biajiz, M. - "Representação de Modelos de Dados Orientados a Objetos através de Parametrização de Abstrações", Tese de Doutorado, São Carlos - SP, Setembro de 1996.
- [Elmasri_1994] Elmasri, R.; Navathe, S. B. - "Fundamentals of Database Systems", Addison-Wesley Publishing Company, 1994, Second Edition.
- [Ferreira_1996] Ferreira, J. E.; Traina, Jr, C. - "Controle de Compartilhamento e Acesso em GBDOO Baseado em Composição de Objetos", XI Simpósio Brasileiro de Banco de Dados, pp. 143 - 157, São Carlos, Outubro 1996.
- [Traina_1997] Traina Jr, C.; Ferreira, J. E.; Biajiz, M. - "Use of a Semantically Grained Database System for Distribution and Control within Design Environments", Lecture Notes on Computer Science nro 1300, pp. 1130 - 1135, Alemanha, Agosto de 1997.
- [Traina_1996] Traina Jr, C.; Traina A. - "Algoritmo para Criação e Gerenciamento de Identificadores de Objetos em um Gerenciador de Objetos", JAIO - 25^o Jornadas Argentinas de Informática e Investigacion Operativa, pp. 2.41 - 2.52, Septiembre 1996.