A Combinação da Técnica de Orientação a Objetos OOA com a Linguagem Formal de Especificação SDL para o Desenvolvimento de um Sistema de Banco de Dados

Nadia Adel Nassif - Walter da Cunha Borelli
Departamento de Telemática - Faculdade de Engenharia Elétrica - Unicamp
Caixa Postal 6101 - 13081-970 - Campinas - SP - Brasil
E-mail : *nadia@dt.fee.unicamp.br - borelli@dt.fee.unicamp.br*

Resumo : Este artigo propõe a utilização da linguagem formal de especificação, SDL (*Specification Description Language*), associada a técnica de orientação a objetos OOA (*Object Oriented Analysis*), como uma metodologia alternativa ao desenvolvimento do sistema de banco de dados SCO (Sistema de Controle Orçamentário) originalmente implementado em SqlWindows. Partindo-se do modelo de objetos do SCO, o sistema é especificado em SDL'92, simulado e validado utilizando-se o pacote SDT (*SDL Design Tool*). O objetivo da aliança OOA e SDL é facilitar as fases de análise e desenvolvimento, e eventual evolução do sistema SCO, de maneira mais eficiente do que a utilização de apenas uma única metodologia.

1. Introdução

Uma das maiores preocupações dos analistas e desenvolvedores de software, é a necessidade de se criar e modificar eficientemente sistemas mantendo-se a confiabilidade. É preciso utilizar técnicas avançadas em engenharia de software que auxiliem no desenvolvimento, análise e projeto de sistemas.

As técnicas orientadas a objetos atendem a algumas destas necessidades. Elas permitem que softwares sejam construídos de objetos que tenham um comportamento especificado. A análise de sistemas no mundo orientado a objetos é feita analisando-se os objetos e os eventos que interagem com esses objetos. O projeto de software é feito reusando-se classes de objetos existentes e, quando necessário, construindo-se novas classes.

Entretanto, as técnicas de orientação a objetos sozinhas, não podem oferecer a magnitude das mudanças tão desejadas no desenvolvimento de software. Para tornarem-se ferramentas muito poderosas, devem ser combinadas a outras técnicas de software como ferramentas CASE e linguagens formais.

O SqlWindows é um ambiente para o desenvolvimento de aplicações cliente/servidor que traz as características e facilidades da orientação a objetos. O Sistema de Controle Orçamentário (SCO)[1] foi implementado em SqlWindows e foi modelado segundo a metodologia de análise e projeto orientado a objetos (OOA)[2].

A linguagem de especificação SDL (*Specification Description Language*) é uma linguagem formal, padronizada pelo CCITT (agora ITU-T), que foi criada originalmente na década de 70, para a especificação de sistemas de telecomunicações. A mais recente recomendação da linguagem SDL, SDL'92 [5], possui características como a utilização de conceitos orientados a objetos, que permitem que a linguagem seja utilizada para a especificação de outros sistemas.

Neste artigo é proposto o uso da linguagem SDL combinada com a descrição do SCO em OOA, como metologia alternativa para seu desenvolvimento. Utilizando-se o SDT[1] (*SDL Design Tool*) a especificação do sistema SCO em SDL pode ser simulada e validada. Através da simulação e validação do SCO especificado em SDL, soluções alternativas podem ser testadas e falhas do sistema podem ser detectadas. Se uma especificação precisa ser alterada, ou deseja-se construir uma evolução do sistema, o esforço necessário para a implementação destas modificações em SDL, é menor do que se o sistema SCO fosse desenvolvido somente usando-se SqlWindows.

2. O Sistema de Controle Orçamentário (SCO) e o Ambiente SqlWindows [1]

O SCO é um sistema de banco de dados monousuário de controle orçamentário desenvolvido para atender às necessidades dos usuários da Coordenadoria de Pesquisa e Pós-Graduação da Universidade Estadual de Londrina.

[1]Pacote SDT, versão 3.02 (*SDT Base, MSC Editor, Validator and Simulator*) : adquirido pelo DT.FEEC/UNICAMP através de um Projeto Temático FAPESP (Proc. 91/3660-0).

Procurando-se utilizar técnicas que melhorassem o desempenho no desenvolvimento e manutenção do software, optou-se por utilizar as técnicas de análise e projeto orientados a objetos [2].

Usando-se conceitos de orientação a objetos, o sistema foi modelado em classes visuais e funcionais. As classes visuais, são responsáveis pela interface com o usuário, isto é, classes de janelas foram construídas para padronizar o sistema. As classes funcionais representam os objetos funcionais com seus atributos e métodos que gerenciam o acesso aos dados do sistema. A figura 1 mostra o diagrama objeto-relacionamento criado na fase de análise do sistema SCO. No diagrama objeto-relacionamento, os retângulos representam tipos de objetos ou classes, e as linhas entre os retângulos, as relações existentes entre os objetos ou classes.

DIAGRAMA OBJETO - RELACIONAMENTO

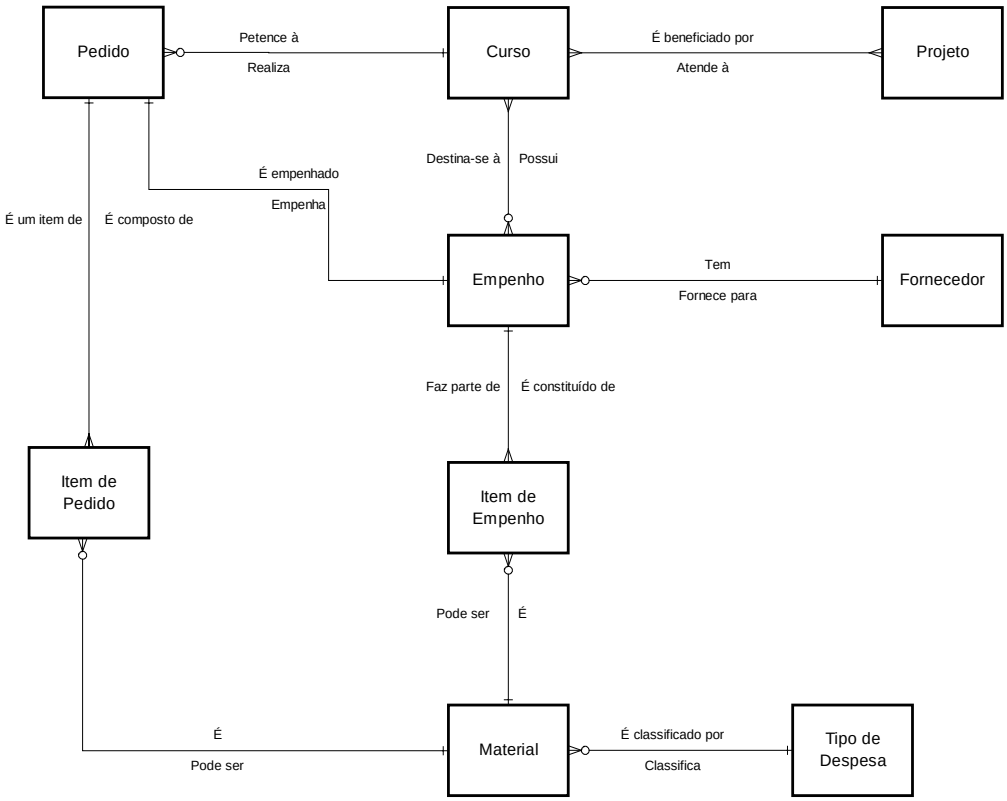


Figura 1 - Diagrama Objeto-Relacionamento do Sistema de Controle Orçamentário (SCO)[1]

Para a implementação do sistema SCO (Figura 1) com modelagem OOA [2], escolheu-se o ambiente SqlWindows que ofere condições para o uso dos conceitos de orientação a objetos. O SqlWindows é um ambiente para o desenvolvimento de aplicações cliente/servidor que podem estar conectadas a diversos tipos de banco de dados. Com sua própria linguagem de programação, SAL (*SqlWindows Application Language*), o desenvolvedor provê de todas as facilidades poderosas da programação orientada a objetos. Além disso, aplicações SqlWindows podem gerenciar uma larga faixa de banco de dados com conexão simultânea, acesso fácil e consistente a dados locais ou em rede. A base de dados do SCO foi implantada em um banco de dados Gupta Corporation de alta performance, *SqlBase* [7], que é um banco de dados local e relacional. O acesso aos

dados é fácil e segue os padrões SQL de buscas e, os dados, são organizados em tabelas relacionais.

3. Modelagem OOA e a Especificação em SDL

O sistema SCO é um sistema de banco de dados que consulta várias tabelas e que as atualiza frequentemente. O modelo de banco de dados escolhido para a sua implementação foi o modelo relacional devido a sua simplicidade e facilidade de acesso. Os dados das tabelas são acessados por métodos declarados em classes, isto é, eles são atributos de classes.

Neste artigo, com o objetivo de mostrar-se o uso combinado da metodologia OOA com a especificação formal em SDL para o desenvolvimento e eventual evolução do sistema SCO, optou-se por se especificar em SDL as duas classes, Curso e Pedido (Figura 1), por serem as principais classes do sistema e, as que realizam mais operações nos dados do banco de dados. Se as outras classes fossem especificadas em SDL, a aplicação da combinação de SDL com a metodologia OOA para o sistema completo, seria de modo análogo.

Na figura 2 a classe Curso é descrita com seus atributos (dados) e métodos. A classe Curso é responsável pelo gerenciamento da tabela de curso. Esta classe representa todos os cursos de pós-graduação existentes na universidade e seus dados.

Os métodos tanto da classe Curso como da classe de Pedido, foram especificados em SDL. A classe de Pedido é responsável por gerenciar os dados da tabela pedido. Um pedido é o documento que oficializa a compra de um material ou equipamento pela universidade, e os pedidos são referentes a um curso.

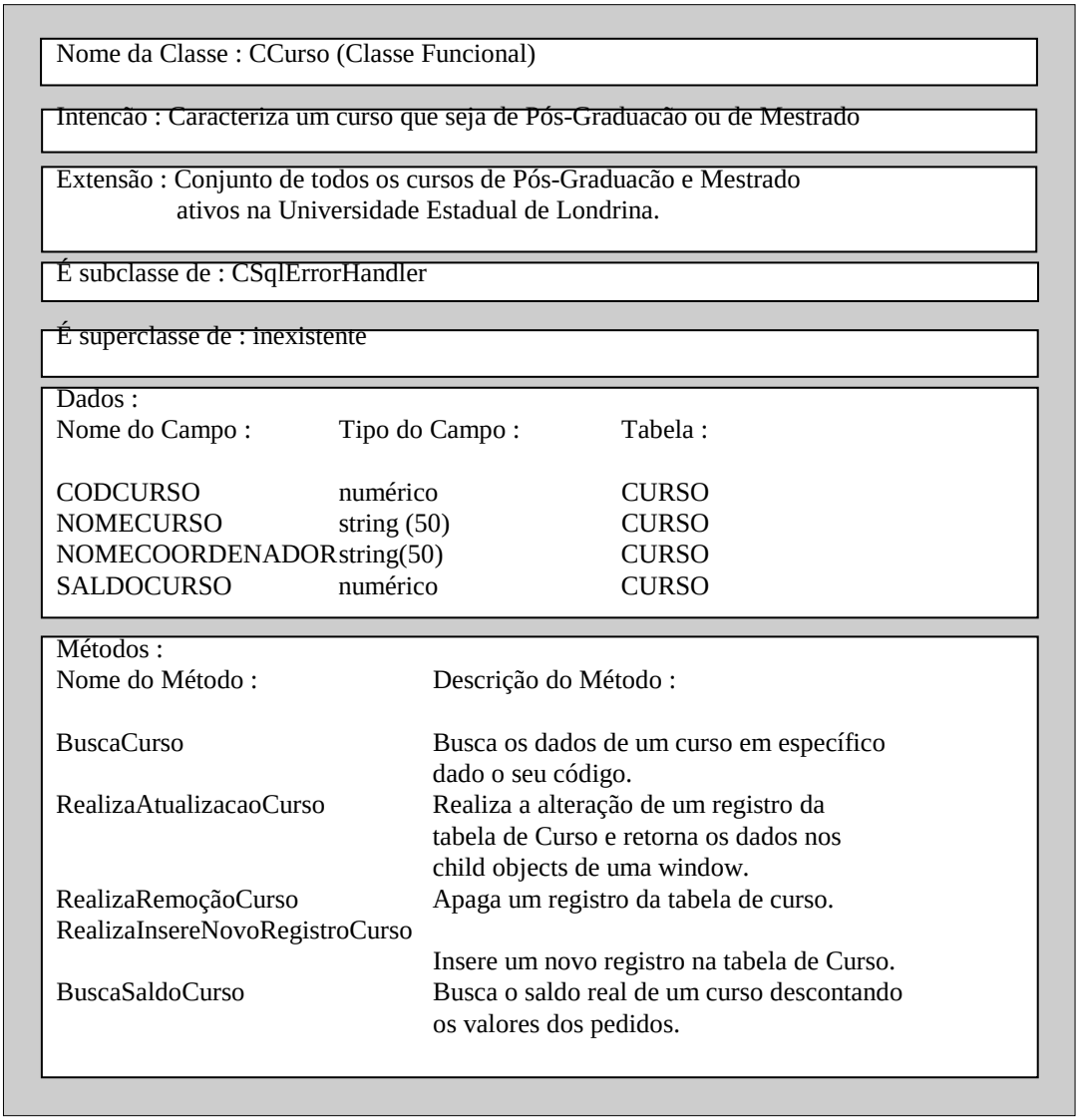
O mapeamento da passagem do modelo do SCO em OOA para SDL [3] são descritas nas tabelas 1 e 2. Em SDL , o sistema é formado por blocos que podem comunicar-se entre si. Desta forma, as duas classes escolhidas são representadas como blocos de nome *Block Course* e *Block Request* respectivamente (Figura 3). Os blocos são formados por processos que podem comunicar-se entre si ou com o ambiente externo (*environment*).

Classe do SCO	Representação em SDL
CCurso	*Block Course*
CPedido	*Block Request*

Tabela 1 - Classes OOA e sua representação em SDL

Os processos da representação em SDL (Tabela 2) exercem as mesmas funções dos métodos da classe Curso do sistema SCO (Figura 2).

Os processos do bloco *Request* especificados em SDL também exercem as mesmas funções dos métodos da classe Pedido do sistema SCO. Os métodos da classe Pedido são métodos similares aos métodos da classe curso como : inserção, remoção e alteração de registros, porém sobre a tabela de pedidos.



Nome da Classe : CCurso (Classe Funcional)

Intencão : Caracteriza um curso que seja de Pós-Graduacão ou de Mestrado

Extensão : Conjunto de todos os cursos de Pós-Graduacão e Mestrado ativos na Universidade Estadual de Londrina.

É subclasse de : CSqlErrorHandler

É superclasse de : inexistente

Dados :

Nome do Campo :	Tipo do Campo :	Tabela :
CODCURSO	numérico	CURSO
NOMECURSO	string (50)	CURSO
NOMECOORDENADOR	string(50)	CURSO
SALDOCURSO	numérico	CURSO

Métodos :

Nome do Método :	Descrição do Método :
BuscaCurso	Busca os dados de um curso em específico dado o seu código.
RealizaAtualizacaoCurso	Realiza a alteração de um registro da tabela de Curso e retorna os dados nos child objects de uma window.
RealizaRemoçãoCurso	Apaga um registro da tabela de curso.
RealizaInsereNovoRegistroCurso	Insere um novo registro na tabela de Curso.
BuscaSaldoCurso	Busca o saldo real de um curso descontando os valores dos pedidos.

Figura 2 - Descrição da Classe de Curso do Sistema SCO[1]

Métodos da Classe Curso	**Representação em SDL**
BuscaCurso	*Process Consult_Record*
RealizaInsereNovoRegistroCurso	*Process Update_Course*
RealizaRemoçãoCurso	*Process Delete_Cascade*
BuscaSaldoCurso	*Process Cash_Cons*
RealizaAtualizaçãoCurso	*Process Update_Course*

Tabela 2 - Métodos da Classe Curso especificadas e suas representações em SDL.

A figura 3 mostra o diagrama do sistema SCO composto dos blocos *Course* e *Request* associados as classes Curso e Pedido do modelo OOA (Figura 1). Os dois blocos comunicam-se entre si através dos canais (*channels*) *Check_Value* e *Cascade.* A comunicação do ambiente externo com os blocos *Course* e *Request* se dão através dos canais *User_Action_Course* e *User_Action_Request* respectivamente.

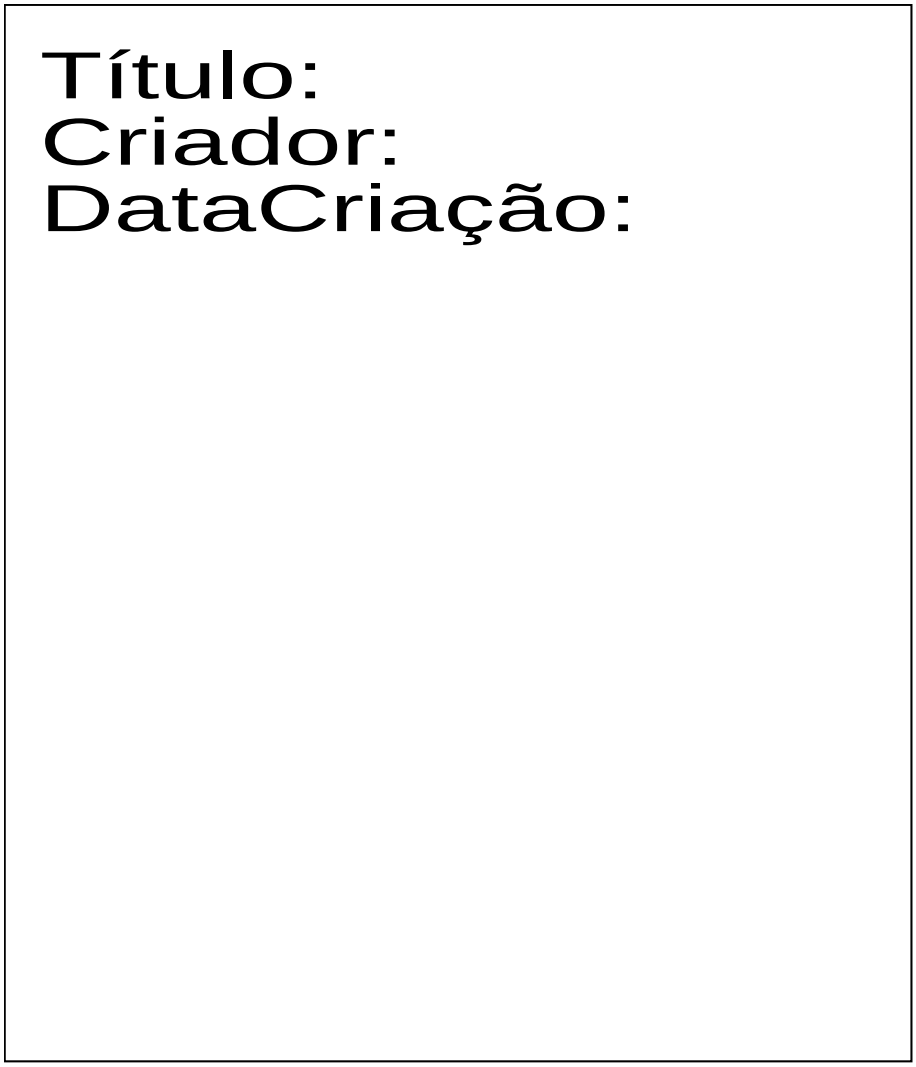


Figura 3 - Sistema SCO em SDL

A figura 4 mostra o diagrama do bloco *Course* com seus processos e a comunicação entre eles. Os processos comunicam-se entre si, com o ambiente e ainda com outro bloco, o bloco *Request,* através de rotas (*routes*).

As tabelas criadas no banco de dados do SCO são representadas por estruturas (*arrays*) em SDL. Estas estruturas armazenam dados fornecidos pelo usuário na simulação e são gerenciadas por um processo específico, que é o processo gerenciador da tabela. Os processos gerenciadores de tabelas na especificação do SCO são : o processo *Table_Manager* no bloco *Course* gerenciando a tabela de curso, e o processo *TableR_Manager* no bloco *Request* gerenciando a tabela de pedido.

Os blocos e processos trocam sinais e informações entre si, porém, somente o processo gerenciador dos dados do bloco pode manipulá-los. Assim como nas classes somente os seus métodos podem realizar operações sobre os seus atributos. Se um processo do *Block Course* deseja consultar ou alterar um dado na tabela de curso, ele envia um sinal com as informações necessárias a realização da operação ao processo *Table_Manager*. Este processo por sua vez, tem como tarefa realizar a operação e retornar ao processo que originou o pedido, um *status* dizendo se a operação foi completada com sucesso.

Figura 4 - *Block Course*

Na figura 4, os processos *Consult_Record, Update_Course* e *Cash_Cons* são processos que realizam consultas aos dados da tabela curso. Desta forma, cada um dos processos possui uma comunicação com o processo *Table_Manager* através de rotas. O processo *Consult_Record* através da rota *RC_T;* o processo *Cash_Cons* através da rota *RCash_Table* e o processo *Update_Course* através das rotas *Man1* e *Man2*. Os processos *Cash_Cons* e *Table_Manager* comunicam-se com o bloco *Request* através dos canais *Check_Value* e *Cascade,* que tornam-se rotas internas ao bloco *Course* de nome *Calc* e *R_DRC* respectivamente.

O processo *Verify_Op* tem uma comunicação com o ambiente externo através do canal *User_Action_Course* (rota *From_User).* Este processo é responsável por verificar a opção de operação que o usuário deseja realizar e então enviar um sinal ao processo que a executará. Por este motivo, o processo *Verify_Op* comunica-se com todos os outros processos com exceção do *Table_Manager*.

Figura 5 - *Block Request*

Da mesma forma no bloco *Request* o processo *TableR_Manager* é o responsável pelo gerenciamento da tabela de pedidos (Figura 5).

Os processos que necessitam acessar dados dos pedidos comunicam-se com o processo gerenciador da tabela de pedidos *TableR_Manager.* O processo *Consult_Req* consulta um registro de pedido e comunica-se com o processo *TableR_Manager* através da rota *R_CT.* O processo *Update_Req* atualiza ou insere um novo registro e comunica-se com o processo *TableR_Manager* através das rotas *R_UT* e *R_UT2*. E o processo *Compute_Total* soma o valor de todos os pedidos relacionados a um curso, e comunica-se com o *TableR_Manager* através da rota *RCp_T*.

O bloco *Request* através do canal *Cascade* comunica-se com o bloco *Course* (Figura 3). Este canal transmite as informações necessárias para a remoção em cascata.

O processo Verify_Req (Figura 5) tem a mesma função do processo *Verify_Op* do bloco *Course* (Figura 4) e comunica-se com os processos *Consult_Req* e *Update_Req* através das rotas *R_VC* e *R_VU* respectivamente.

Para exemplificar a especificação de um processo em SDL, o processo *Table_Manager* do bloco *Course* é mostrado nas figuras 6, 7 e 8. Neste processo, as operações de consulta, alteração, inserção e remoção de um registro da tabela de curso são implementadas.

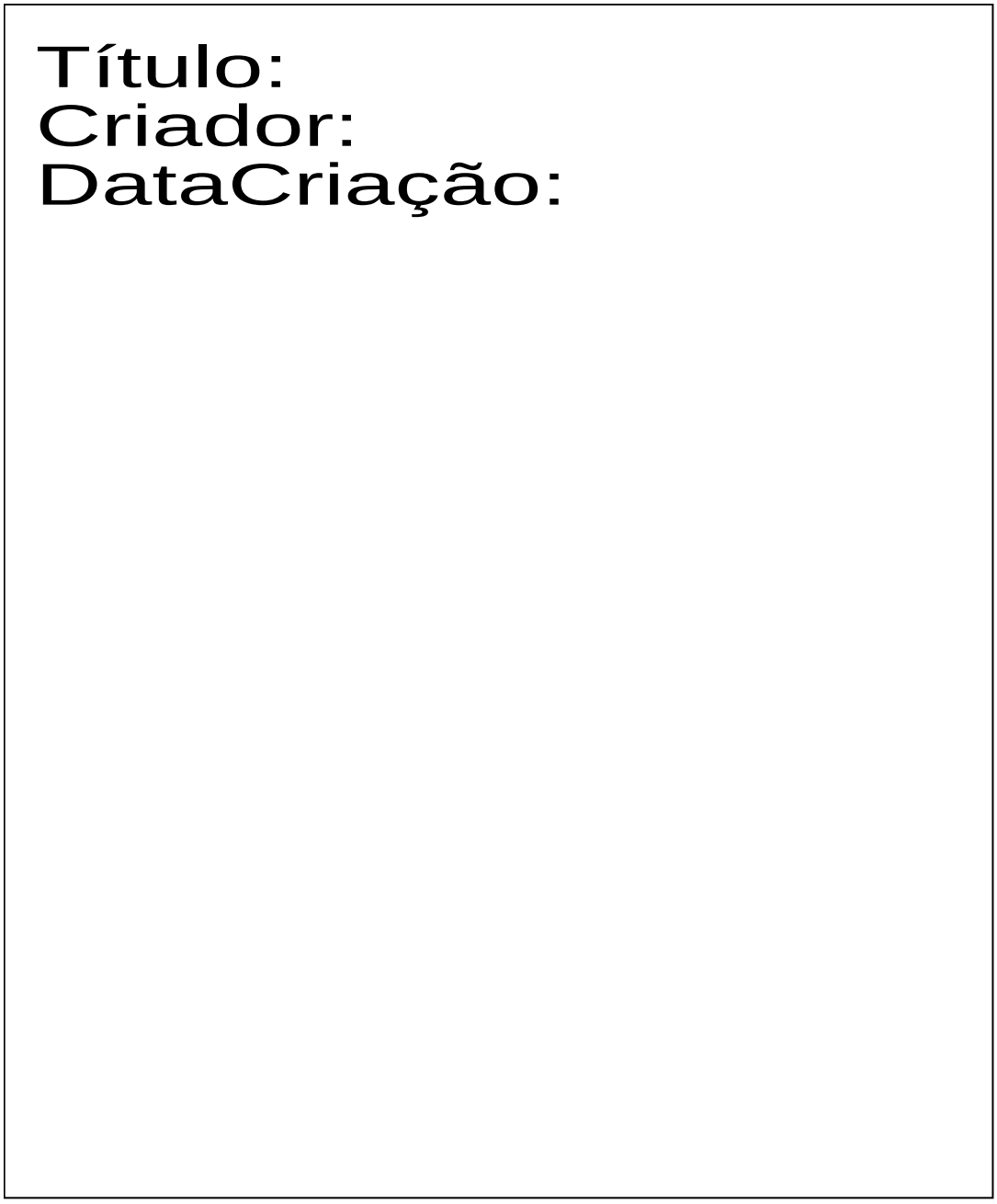


Figura 6 - *Process Table_Manager (1 de 3)*

Na figura 6, o processo *Table_Manager* fica no estado *Waiting* até que receba o sinal *Op_Size3* ou *Op_Size2* provenientes dos processos *Consult_Record* e *Cash_Cons* respectivamente (Figuras 4 e 6). Percorrendo a tabela *Real_CTable,* o processo procura pelo registro que possua o código do curso igual ao código recebido *C_Key.* Se o registro for encontrado, (*Message_C := Record Found)*, o sinal *Size3* é enviado ao processo *Consult_Record,* ou o sinal *Size2* é enviado ao processo *Cash_Cons*. Os dois sinais, *Size3* e o *Size2,* contêm dois parâmetros que são o registro encontrado (*Real_CTable(cont))* e a mensagem "*Record Found".*

Caso o registro não seja encontrado, os mesmos sinais são enviados com os parâmetros contendo um registro nulo e a mensagem *"Record Not Found".*

Na figura 7, o processo *Table_Manager* permanece no estado *Waiting* até que receba os sinais *Op_Size1* ou *New_C* provenientes do processo *Update_Course.* Quando o sinal *Op_Size1* é recebido, o processo *Table_Manager* procura pelo registro *Record* na tabela *Real_CTable.* Se o registro for encontrado, ele será atualizado com os dados enviados no parâmetro *Record* e o sinal *Size1* é enviado ao processo *Update_Course.* O parâmetro de valor 1 do sinal *Size1* indica que o registro foi atualizado, e o valor 2, que o registro não foi encontrado.

Quando receber o sinal *New_C* o processo *Table_Manager* inserirá um novo registro de curso na tabela de curso *RealC_Table.* O tamanho da tabela é incrementado de uma unidade *Table_Size,* e o sinal *New_C_Result* é enviado ao processo *Update_Course* contendo um parâmetro com a mensagem *"New Record Inserted".*

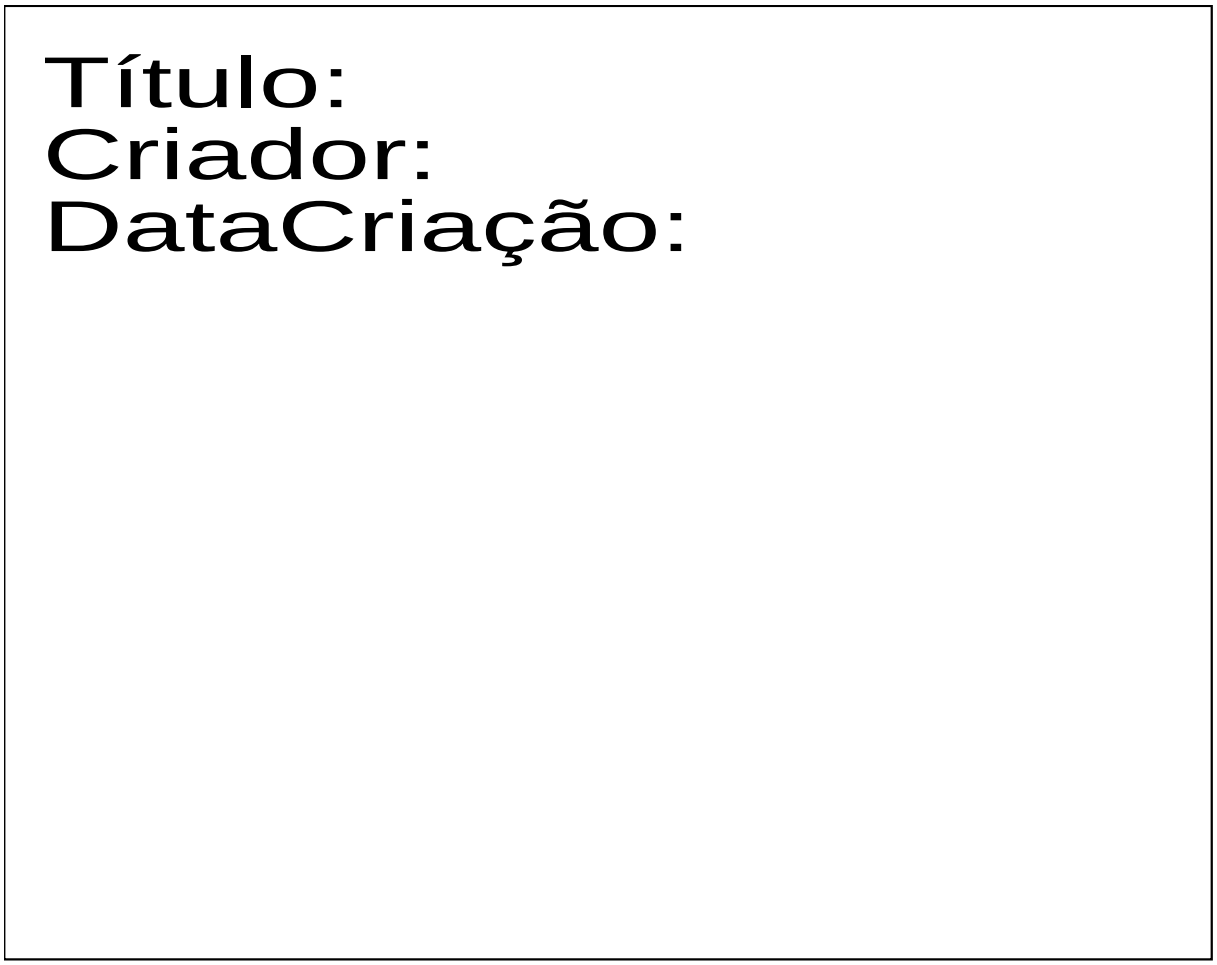


Figura 7 - *Process Table_Manager* (2 de 3)

Na figura 8, é apresentado a implementação da remoção em cascata. O processo *Table_Manager* permanece no estado *Waiting* até que receba o sinal *Op_Size4* proveniente do processo *Delete_Cascade* (Figura 4).

O registro de código *C_Key* é procurado na tabela *RealC_Table.* Caso não seja encontrado, o sinal *Size4* com a mensagem *"Record Not Found"* é enviado ao processo *Delete_Cascade.*

Figura 8 - *Table_Manager (3 de 3)*

Caso o registro seja encontrado, um *shift* de uma posição a esquerda é feita na tabela *RealC_Table* da posição *cont* até o fim da tabela (condição para o loop que realiza o *shift* na tabela : *Control_D <= Table_Size - contaux,* Figura 8). Terminado o *shift* a esquerda, o tamanho da tabela é decrementado de uma unidade (*Table_Size := Table_Size - 1*) e o sinal *Req_Del_Cascade* é enviado ao bloco *Request* que será responsável por remover todos os pedidos que estejam relacionados ao curso de código *C_Key.*

O processo *Table_Manager* permanece no estado *Deleting_Req* até que receba o sinal *Del_Result* proveniente do bloco *Request* (Figura 3) com o parâmetro que traz a mensagem obtida durante a deleção.

O sinal *Size4* é enviado ao processo *Delete_Cascade* com a mesma mensagem recebida pelo sinal *Del_Result.*

4. Resultados

4.1 Simulação

O modelo de banco de dados do sistema SCO (Figura 1), é um modelo de banco de dados relacional (Figura 9). Desta forma, as tabelas estão relacionadas entre si através de ligações de chaves primárias e estrangeiras. No SCO, a tabela de pedido está ligada a tabela de curso. A tabela de curso possui uma chave primária que especifica um curso somente. Cada pedido está relacionado a somente um curso, portanto, a chave primária da tabela curso, é uma chave estrangeira na tabela pedido de acordo com o modelo relacional.

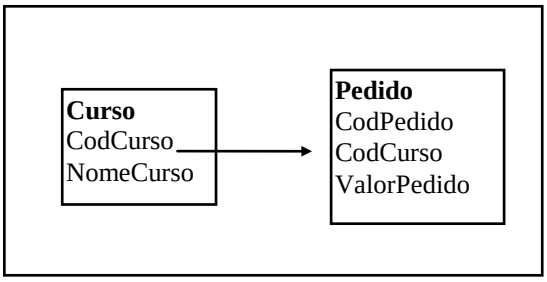


Figura 9 - Relacionamento de Tabelas do SCO

No sistema SCO a remoção de um curso foi implementada como remoção restrita [1] , isto é, se um curso é apagado, os pedidos relacionados a este curso não são apagados ao menos que o usuário solicite a remoção .

Especificando-se as duas tabelas em SDL, a simulação de como o sistema se comportaria se a remoção de um curso fosse implementada como remoção em cascata pode ser testada usando-se o *Simulator* do SDT [4]. Na remoção em cascata se um curso é removido, todos os pedidos relacionados a ele são removidos automaticamente. O SDT (*SDL Design Tool*), que é um ambiente para o desenvolvimento de simulações e validações de especificações SDL, proporcionou um teste confiável e rápido para uma implementação alternativa do sistema SCO.

Na figura 10, pode-se visualizar uma parte da simulação no MSC (*Message Sequence Chart*) [6] com a obtenção de sucesso numa operação de remoção em cascata.

O usuário envia o sinal *Course_Op (env_1, environment)* com os parâmetros (4, (.1,0.)). O valor 4 indica a operação remoção em cascata definido na especificação. O segundo parâmetro é uma estrutura que traz os atributos do curso. O valor 1 indica que o curso de código número 1 deve ser apagado, e o segundo parâmetro é nulo (dados do curso) pois só é necessário conhecer-se o código para realizar a remoção.

O sinal *Course_Op* é enviado do *environment* ao processo *Verify_Op* do bloco *Course* (Figura 4). O processo *Verify_Op* identifica a operação como remoção e envia o sinal *Del_C(1)*, onde o valor 1 é o número do código do curso, ao processo *Delete_Cascade* e permanece no estado *Deleting_C.*

O processo *Delete_Cascade* envia o sinal *Op_Size4(1),* sendo 1 o número do curso, ao processo *Table_Manager* e permanece no estado *Deleting_C.* O processo *Table_Manager* que é o responsável por atualizações na tabela de curso, remove o curso de número 1 e envia o sinal *Req_Del_Cascade(1)* (Figura 8) para o processo gerenciador da tabela de pedidos do bloco *Request, TableR_Manager,* permanecendo no estado *Deleting_Req*.

O processo *Table_R_Manager* apaga todos os registros de pedido que estavam relacionados ao curso 1 e envia o sinal *Del_Result* de volta ao processo *Table_Manager* (Figura 8) do bloco *Course* com a mensagem *" All Records Related to Course Deleted"* e permanece no estado *Waiting.*

O processo *Table_Manager* envia o sinal *Size4* com a mesma mensagem como parâmetro do sinal ao processo *Delete_Cascade.*

O símbolo * indica que até este ponto da simulação o sinal *Size4* ainda não tinha sido consumido pelo processo *Delete_Cascade.* Os sinais continuam sendo mandados de processos a processos até que o usuário (*environment*) receba a mensagem obtida na remoção em cascata. Entretanto a Figura 10 apresenta apenas uma pequena parte da simulação como exemplificação.

Outras simulações foram realizadas com repetidas operações como inserir um curso de mesmo código duas vezes. O sistema respondeu corretamente atualizando o registro e não duplicando. Simulações de consulta do saldo de um curso também tiveram sucesso. Além de consultar o saldo do curso, o sistema procurou por todos os pedidos relacionados ao curso que contavam como débito em curso, e, retornou o saldo atualizado.

4.2 Falhas na Especificação do SCO detectadas na Simulação

Durante a simulação verificou-se que uma parte do sistema encontrava-se em "*loop*" e que um sinal estava sendo mandado para um processo que não estava preparado para recebê-lo. Esses tipos de falhas de especificação do sistema SCO detectados durante várias simulações foram corrigidos retornando-se aos diagramas de blocos, e destes aos processos que apresentavam problemas não havendo a necessidade de alteração no correspondente modelo de objetos (Figura 1).

Figura 10 - Simulação da Remoção em Cascata

4.3 Validação

O sistema SCO especificado em SDL (Figura 3) foi validado pelo *Validator* [4] e os resultados obtidos são mostrados na figura 11.

A validação simboliza as várias operações repetidas de um usuário utilizando o sistema durante um período de tempo. Validando-se um sistema é possível identificar falhas

como, a falta de tratamento de situações não previstas pelo programador, ou até mesmo erros na lógica de programação.

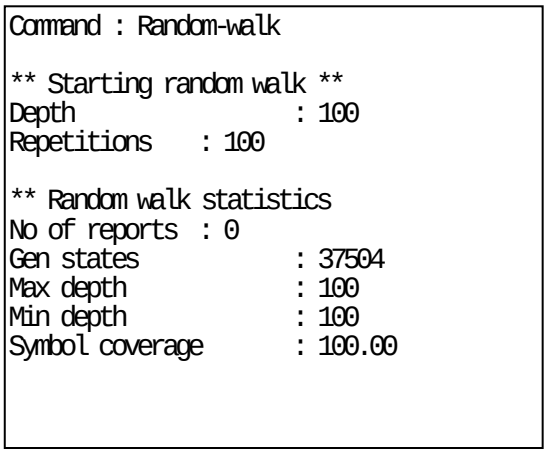


Figura 11 - Resultado da Validação do SCO

Numa repetição de 100 vezes o número de mensagens de falhas detectadas (*reports*) foi igual a zero. Isso significa que nenhuma situação de falha foi encontrada. O sistema SCO apresentava um processo que quando executado entrava em "*loop"*. Entretanto, este erro foi detectado na simulação e corrigido antes de se validar o sistema.

Symbol coverage igual a 100.00 significa que 100% dos símbolos do sistema foram cobertos pela validação.

5. Conclusão

A especificação alternativa do Sistema de Controle Orçamentário (SCO) utilizando-se a linguagem formal SDL combinada com a metodologia OOA, mostrou-se bastante eficaz. Através da simulação foi possível observar o comportamento do sistema diante de operações do usuário, troca de informações entre classes e acesso a dados nas tabelas relacionais.

Soluções alternativas de gerenciamento de dados foram testadas, como a remoção em cascata, de maneira muito mais rápida e eficiente do que se elas fossem implementadas em SqlWindows, que exigiria uma reestruturação das classes envolvidas e dos comandos SQL de acesso ao banco de dados.

Durante a validação do sistema SCO foi possível constatar através de diagramas MSC (*Message Sequence Chart*), o funcionamento dos principais blocos do sistema diante de repetidas operações do usuário.

O uso de uma linguagem formal (SDL'92) aliada às técnicas de orientação a objetos (OOA) e a ferramentas de implementação de sistemas de banco de dados (SqlWindows), é de grande utilidade ao desenvolvedor que precisa testar alterações e comportamentos de sistemas. Além da validação e simulação do sistema através do SDT (*SDL Design Tool*), a especificação em SDL, permite facilidades para uma eventual evolução do sistema SCO com a utilização de recursos e conceitos de orientação a objetos como o reuso e herança, e através de construções em SDL'92, como *packages, block types, process types , virtual* e *redefined blocks.*

6. Referências

[1] - Nassif, Nadia A. , Sistema de Controle Orçamentário, Documentação do Sistema de Controle Orçamentário (SCO), Centro de Ciências Exatas, Departamento de Computação da Universidade Estadual de Londrina, 1996.

[2] - MARTIN, James; J. James, Análise e Projeto Orientados a Objetos, Makron Books, São Paulo : Milton Mira de Assunção Filho, 1995.

[3] - Belina, F. Hogrefe D., Sarma, A., *SDL with Applications from Protocol Specification,* Prentic Hall, 1991.

[4] - Telelogic AB (Malmo, Sweden). *Getting Started with SDT3.02,* Malmo, 1995.

[5] - ITU-T(Geneve, Switzerland). *Recommendation Z.100 CCITT Specification and Description Language (SDL) : programming languages.* Geneve, 1993.

[6] - ITU-T (Geneve, Switzerland). *Recommendation Z.120. Message Sequence* Chart (MSC). Genve,1993.

[7] - *SqlBase SQL Reference,* SqlBase, versão 5.0, Gupta Tecnologies, Inc., 1994.