Modelagem Orientada ao Objeto de Especificações Escritas em Estelle (*)

Ciro de Barros Barbosa (**) , Wanderley Lopes de Souza

Grupo de Sistemas Distribuídos e Redes (GSDR)
Departamento de Computação (DC)
Universidade Federal de São Carlos (UFSCar)
Cx. 676, 13565-905 São Carlos (SP)
E-mail: pcbb@iris.ufscar.br, dwls@power.ufscar.br

Resumo

A Técnica de Descrição Formal (TDF) *Extended State Transition Language (Estelle),* padronizada pela *International Organization for Standardization (ISO),* foi desenvolvida para a especificação formal de Sistemas Distribuídos (SD) e protocolos de comunicação. A partir da especificação formal de um SD é possível derivar semi-automaticamente a implementação desse sistema. O objetivo principal deste artigo é o de propor um mapeamento dos conceitos arquitetônicos da TDF Estelle nos elementos próprios às linguagens orientadas ao objeto. Este trabalho deverá servir de subsídio para o desenvolvimento de ferramentas, que visem a geração, a partir de especificações Estelle, de implementações em linguagens de programação orientadas ao objeto.

Abstract

The Formal Description Technique (FDT) *Extended State Transition Language (Estelle),* a standard by *International Organization for Standardization (ISO),* was developed for the formal specification of Distributed Systems (DS) and communication protocols. It is possible to derive a DS semi-automatic implementation from its formal specification. The main goal of this paper is to suggest a mapping of the Estelle architectural concepts onto the standard features of the object-oriented languages. This work will be used as a basis for the development of tools that generate implementations in object-oriented programming languages from Estelle formal specifications.

1. Introdução

Durante o ciclo de desenvolvimento (especificação, verificação, implementação e teste) de um Sistema Distribuído (SD), é possível a utilização de ferramentas para a geração semi-automática de implementações desse sistema a partir de sua especificação formal, desde que esta seja realizada utilizando-se uma Técnica de Descrição Formal (TDF).

(*) realizado com o auxílio do CNPq e da CAPES
(**) mestrando junto ao PPGCC da UFSCar

Um primeiro passo para a construção de tais ferramentas é a concepção de uma estrutura genérica de implementação, que suporte os recursos da semântica da TDF utilizada e que possa ser adequada às peculiaridades de cada especificação.

Através do estudo de algumas dessas ferramentas [GERB 83, JARD 86, EWS 89, DOLD 92], observa-se que o grau de complementação manual do código gerado é proporcional à diferença entre os poderes de abstração das linguagens de especificação e de implementação do sistema [LIN 92]. Para diminuir essa diferença e, consequentemente, esse grau de complementação manual, deve-se aumentar o poder de abstração da linguagem de implementação. O sentido inverso, isto é, a diminuição do poder de abstração da linguagem de especificação, não é desejável, uma vez que abstração é uma característica fundamental na fase de especificação de sistemas complexos.

Este trabalho apresenta um mapeamento dos conceitos arquitetônicos da TDF *Extended State Transition Language (Estelle)* [ISO 88] nos elementos próprios às linguagens orientadas ao objeto. A utilização desse paradigma aumenta o poder de abstração a nível de implementação, diminuindo significativamente o grau de complementação manual no processo de geração automática do código de implementação.

Alguns detalhes de implementação peculiares aos SDs são abstraídos pela linguagem de especificação. Por exemplo, o tratamento da comunicação entre módulos de um mesmo SD, que são executados em diferentes Sistemas Operacionais, e o controle do sincronismo de execução entre os mesmos. Consequentemente, serão introduzidos alguns elementos nesse mapeamento que não possuem uma correspondência a nível de especificação.

Conjugando-se esse mapeamento com a construção de bibliotecas de classes, tem-se uma boa base para o desenvolvimento de ferramentas, que visem a geração automática de código de implementação de SDs com grau de complementação manual bastante reduzido.

2. Estelle e Programação Orientada ao Objeto (POO)

Em Estelle uma especificação é composta de um conjunto de módulos. Um módulo pode ser refinado em submódulos, definindo um parentesco e uma estrutura hierárquica entre os componentes da especificação. O comportamento de um módulo é descrito por uma Máquina de Estados Finita Estendida (MEFE).

Cada módulo é representado por uma caixa preta com portas de entrada/saída, denominadas pontos de interação. A cada ponto de interação é associada uma fila de comprimento infinito, que é utilizada para armazenar as mensagens recebidas por aquele ponto. Canais de comunicação bidirecionais podem conectar pontos de interação, sendo que estes definem os conjuntos de interações a serem emitidas e recebidas por cada ponto.

Essencialmente a POO envolve a criação de objetos, os quais incluem tanto os dados quanto o código que opera sobre esses dados. Os objetos podem conter elementos de acesso público ou privado. O acesso privado, a um determinado elemento, restringe a manipulação desse

elemento a componentes do próprio objeto, controlando as operações sobre ele. A vantagem da utilização de objetos é que eles são simples entidades lógicas, mais fácil de serem compreendidas e manipuladas.

A POO permite que seja criada uma hierarquia de objetos, possibilitando uma movimentação do objeto mais geral para o mais específico. Nessa hierarquia, cada objeto herda características daqueles que o precederam, o que é conhecido como herança. Outro recurso é o polimorfismo, que permite a um nome ser utilizado como referência para uma classe geral de ações. Dependendo do tipo de dado que está sendo tratado, uma ação específica de uma classe geral de ações é executada.

Outra característica oportuna é o conceito de instância: um objeto representa um tipo de dado e a definição de uma variável associada a esse tipo representa uma instância do objeto correspondente. Estelle utiliza esse mesmo conceito: por exemplo, um módulo representa um tipo e uma variavel associada a esse tipo representa uma instância inativa do módulo correspondente.

Em [DEMB 89, SOUZ 89] são apresentados tutoriais sobre a TDF Estelle, enquanto que [RUMB 91] trata da POO.

3. Notação gráfica utilizada para o mapeamento

A modelagem conceitual da arquitetura é feita utilizando-se a notação gráfica apresentada em [VIEI 93] a qual é resumida na Fig. 1. A estrutura de um objeto é modelada através de uma coleção de componentes (atributos). Cada componente pode possuir um tipo simples, um tipo composto ou um tipo definido. Um tipo simples é expresso através de tipos básicos, tais como: integer, real, char, etc. Um tipo composto é definido através dos seguintes construtores de tipos:

- *Tupla*, para o agrupamento de diferentes componentes;
- *Conjunto*, para expressar a múltipla ocorrência de um componente sem repetição;
- *Lista*, para expressar a múltipla ocorrência ordenada de um componente.

Nesta notação gráfica (Fig. 1), um objeto é representado por um retângulo com o nome do objeto e o mecanismo de herança da POO é expresso por uma seta, que aponta para o tipo base da relação.

As tuplas devem possuir um nome de referência. É possível também que um componente do objeto seja uma referência para outro objeto da modelagem. O grafo apresentado na Fig. 2 mostra as possíveis construções em uma modelagem.

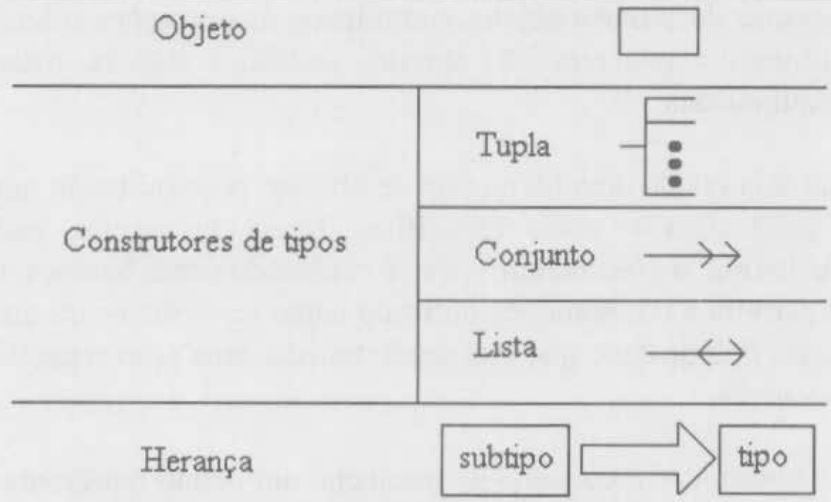

Fig. 1 Notação gráfica usada na modelagem

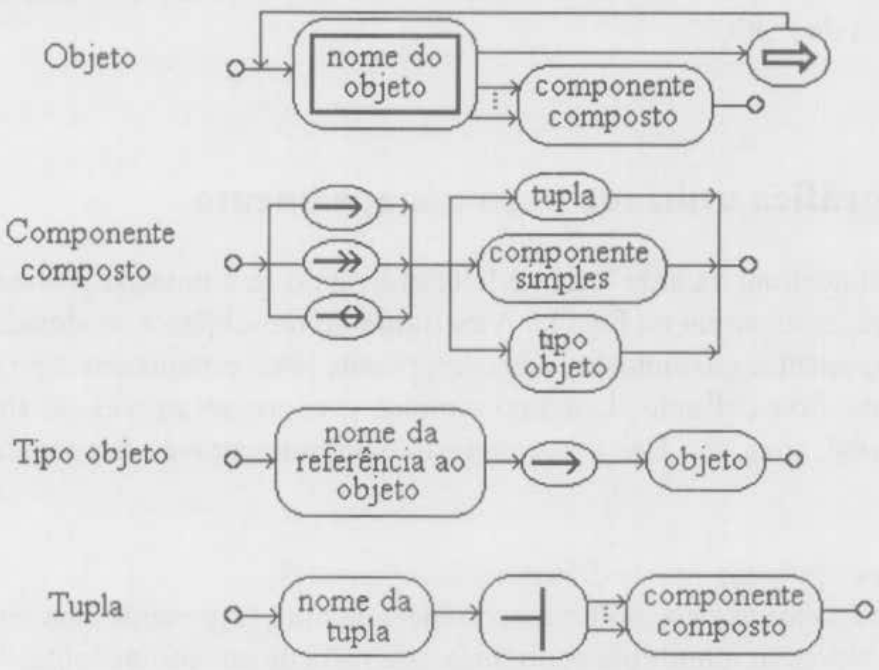

Fig. 2 Grafo sintático da notação gráfica

4. Mapeamento dos conceitos arquitetônicos de Estelle

Para efeito de mapeamento, um conceito arquitetônico de Estelle pode ser dividido em dois tipos de componentes: os estáticos, que são comuns a todas especificações, e os dinâmicos, que dependem da especificação. Cada um desses conceitos deve ser mapeado utilizando-se o mecanismo de herança da POO, para que possa ser construída uma hierarquia de objetos.

O objeto, que se encontra no topo dessa hierarquia, representa os componentes estáticos do conceito arquitetônico e deve capturar o seu comportamento e os seus atributos básicos. Essas características serão herdadas por objetos, que se encontram nos níveis inferiores dessa hierarquia, que representam os componentes dinâmicos do conceito arquitetônico.

4.1. Módulo

Este é o conceito arquitetônico mais importante da TDF Estelle. No módulo estão contidas as definições e inicializações dos elementos que constituem a especificação. Também o comportamento da especificação é expresso no interior dos módulos através de MEFEs, que são basicamente compostas de estados e transições, sendo que cada transição é, por sua vez, basicamente composta de condições e ações. A definição de um módulo é composta de duas partes: um cabeçalho e um corpo, os quais serão associados no momento da inicialização do módulo.

Grande parte dos componentes de um módulo é dependente da especificação. Por exemplo, o conjunto de transições, as variáveis, os procedimentos, os pontos de interação, os sub-módulos definidos num eventual refinamento do módulo, os canais a serem utilizados pelos pontos de interação do módulo, etc... Entretanto, existem componentes de um módulo que são encontrados em qualquer especificação e que podem constituir um objeto base para este conceito arquitetônico. A modelagem desse tipo de objeto é apresentada na Fig. 3.

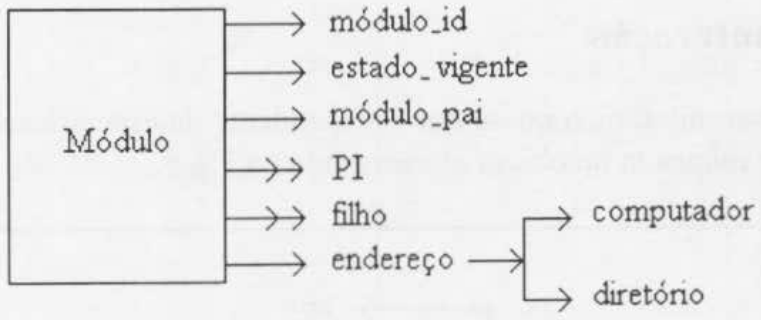

Fig. 3 Modelagem do tipo de objeto *Módulo*

Para percorrer a árvore de módulos, que compõem uma especificação, são necessários os atributos *módulo_id*, *módulo_pai* e *filho*, que são respectivamente uma identificação para o módulo em questão, uma referência ao módulo pai e uma lista referenciando os módulos filhos. Um endereço, com a composição apresentada, é necessário por se tratar de um sistema distribuído possivelmente entre dois ou mais computadores. O *estado_vigente* do módulo e a lista de módulos filhos são atributos pertencentes a qualquer módulo, embora sejam manipulados por métodos em níveis mais baixos na hierarquia de objetos.

A TDF Estelle expressa as restrições de paralelismo e sincronismo, entre a execução de transições de diferentes módulos, através de qualificadores para os módulos: *systemprocess*, *systemactivity*, *process* ou *activity*. Um módulo não qualificado é inativo e serve para auxiliar a estruturação da especificação. Essa diversidade de comportamentos denota uma especialização na funcionalidade do objeto do tipo módulo, o que implica na criação de um sub-tipo desse tipo de objeto para cada comportamento possível. Tal mapeamento é apresentado na Fig. 4.

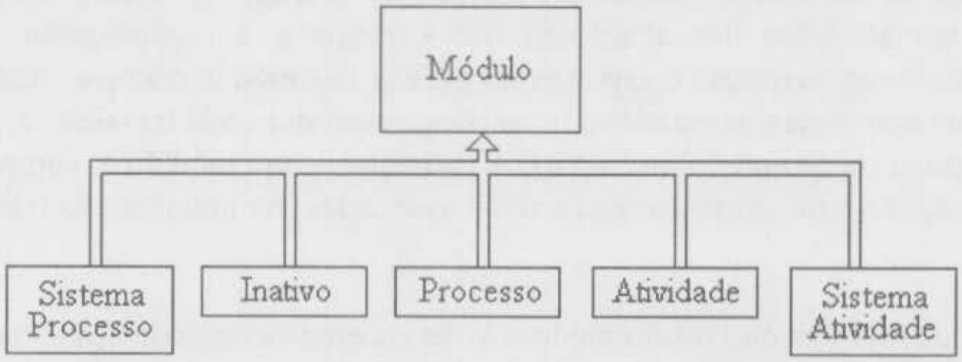

Fig. 4 Modelagem dos sub-tipos do objeto do tipo *Módulo*

Os novos sub-tipos não implicam na inclusão de nenhum atributo, diferindo apenas nos métodos que regulam a execução das transições.

4.2. Ponto de interação

Este é um conceito arquitetônico quase que independente da especificação. Ele possui alguns atributos e pode ser mapeado no objeto representado na Fig 5.

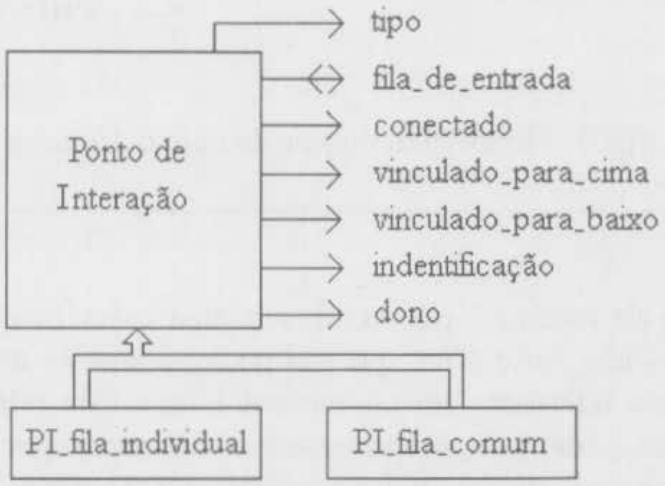

Fig. 5 Modelagem do objeto do tipo *Ponto de Interação*

A fila que armazena as interações, recebidas pelo ponto de interação, pode ser relacionada a um único ponto ou ser comum a diversos pontos. Essa peculiaridade da semântica de Estelle impõe métodos distintos para a mesma função (*armazenar_interação_recebida*). Essa característica é conhecida na POO como polimorfismo e pode ser implementada como sendo métodos de dois sub-tipos do tipo de objeto ponto de interação. Consequentemente, aparecem na modelagem da Fig. 5 dois tipos de objetos: *PI_fila_individual* e *PI_fila_comum*.

Para que um link de comunicação possa ser percorrido de um extremo ao outro, é preciso que cada ponto intermediário possua referências para os pontos eventualmente vinculados a ele e para o ponto eventualmente conectado a ele. Consequentemente, aparecem na modelagem da Fig. 5 os seguintes atributos: *conectado*, *vinculado_para_cima* e *vinculado_para_baixo*. Determinadas operações sobre os pontos de interação necessitam da definição do tipo de ponto (interno ou externo), justificando assim a inclusão do atributo *tipo*. Uma identificação para o ponto e uma referência para o módulo ao qual pertence são necessárias, para que várias instâncias desse mesmo ponto possam ser diferenciadas. Finalmente, a fila contendo as interações recebidas pelo ponto de interação requer o atributo *fila_de_entrada*.

4.3. Interação

Este conceito pode ser mapeado como um tipo de objeto cujos componentes são: identificação de cada uma das diversas instâncias possíveis (*origem* e *destino*), um código identificando o tipo da interação dentro de um conjunto de tipos possíveis (*código*) e uma estrutura que armazena os parâmetros transportados por aquela interação (*dados*). A modelagem desse tipo de objeto é apresentada na Fig. 6.

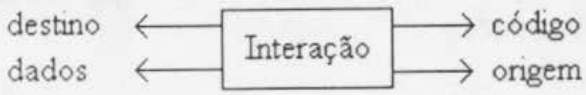

Fig. 6 Modelagem do objeto do tipo *Interação*

4.4. Componentes dinâmicos dos conceitos arquitetônicos

Os objetos até agora apresentados, que se encontram nos topos das hierarquias de objetos estabelecidas e que representam os compontentes estáticos dos conceitos arquitetônicos, juntamente com outros objetos, que serão apresentados em seções posteriores, constituirão uma biblioteca de classes que deverá ser utilizada por qualquer implementação.

Os componentes dinâmicos dos conceitos arquitetônicos serão implementados, em função das peculiaridades da especificação, por objetos que não farão parte da biblioteca de classes (e.g., *Corpo_Módulo*, *Interação_Específica*), mas que se relacionarão com esta a partir das hierarquias de tipos de objetos estabelecidas. A Fig. 7 mostra essas hierarquias para os conceitos arquitetônicos módulo, ponto de interação e interação.

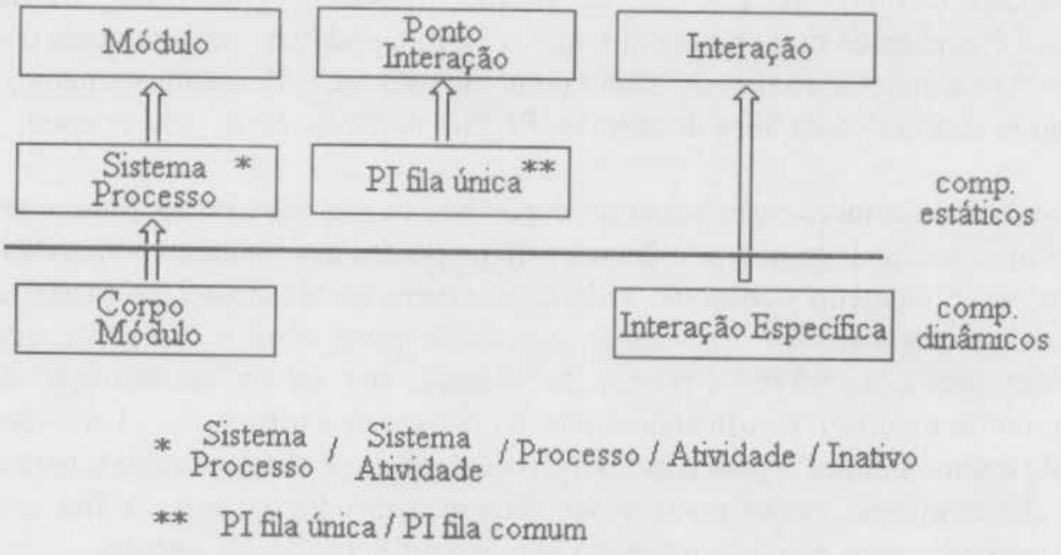

Fig. 7 Hierarquias de tipos de objetos

4.5. Ciclo de execução

Em Estelle cada sub-sistema mantém seu próprio ciclo de execução. Em cada ciclo de execução um conjunto de transições, pertencentes à sub-árvore do sub-sistema em questão, é selecionado e executado de acordo com as regras de paralelismo e sincronismo definidas pela semântica dessa TDF. Portanto, deve existir, a nível de implementação, um mecanismo que controle a execução dessas transições.

Neste trabalho é proposto um mecanismo, baseado na concessão de autorizações, para a execução das transições que deve obedecer às seguintes regras:

- um sub-sistema é autorizado a executar no início de cada ciclo de execução, sendo que um novo ciclo terá início somente após o término da execução de todas as transições relativas ao ciclo precedente;

- um módulo autorizado para execução, que tenha transições disparáveis, executa uma delas;

- um módulo autorizado para execução, que não tenha transições disparáveis, transfere a autorização para todos os seus filhos, se ele for um *systemprocess* ou um *process,* ou para apenas um deles, se ele for um *systemactivity* ou um *activity*, sendo que essa escolha é indeterminística;

- a fim de determinar se qualquer filho ou descendente possui transições disparáveis, um módulo *systemactivity* ou *activity* envia uma requisição de oferta de transição a todos os seus filhos; ao receber essa requisição, um módulo *activity* responderá positivamente se ele ou qualquer um de seus filhos tiver transições para execução.

4.6. Métodos

Os métodos de um objeto são procedimentos que manipulam os dados daquele objeto. Para cada um dos objetos já modelados, são apresentados na Fig. 8 os seus principais métodos. Tais métodos desempenham as atividades do ciclo de execução de um sub-sistema.

Objeto	Método
Módulo	- inicializa os seus atributos; - cada um dos seguintes comandos de Estelle é um procedimento que atua sobre os atributos de um módulo sendo, portanto, um método: output, init, terminate, connect, disconnect, attach, detach e release.
Inativo	- inicializa todos os descendentes e finaliza a execução do módulo.
Sistema Processo	- inicializa o seu ciclo de execução; - executa transição selecionada; - delega para cada módulo filho o direito de execução de uma transição e aguarda o fim da execução das transições de cada módulo filho.
Sistema Atividade	- inicializa o ciclo de execução do módulo; - executa transição selecionada; - envia a cada módulo filho uma requisição de oferta de transição; aguarda as respostas; seleciona, entre os módulos filhos que tenham oferecido uma transição, um para transferir a permissão de execução; transfere a permissão; aguarda o fim da execução.
Atividade	- aguarda pela requisição de oferta de transição ou pela permissão de execução, caso não seja local ao módulo pai; - executa transição selecionada; - sinaliza ao módulo pai o fim da execução da sua transição; - envia a cada módulo filho uma requisição de oferta de transição; aguarda as respostas; seleciona, entre os módulos filhos que tenham oferecido uma transição, um para transferir a permissão de execução; transfere a permissão; aguarda o fim da execução; sinaliza ao módulo pai o fim da execução.
Processo	- aguarda pela permissão de execução, caso não seja local ao módulo pai; - executa transição selecionada; - sinaliza ao módulo pai o fim da execução da sua transição; - transfere a permissão de execução para os módulos filhos; aguarda o fim das execuções; sinaliza ao módulo pai o fim das execuções.

Fig. 8 Tabela de métodos

Objeto	Método
Ponto de Interação	- retorna a primeira interação da *fila_de_entrada*; - exclui a primeira interação da *fila_de_entrada*; - inclui interação na *fila_de_entrada*.
Interação	- atribui destino; - atribui origem; - retorna o código da interação.
Corpo Módulo	- inicializa atributos definidos na especificação.
Interação Específica	- inicializa o atributo *código_da_interação*.

Fig. 8 Tabela de métodos (continuação)

5. Implementação das demais características de Estelle

Algumas características de Estelle não foram contempladas na modelagem apresentada, por não se caracterizarem como objetos. Nas próximas seções, são apresentadas propostas de implementações correspondentes a essas características.

5.1. Transições

Cada transição em Estelle é composta de condições, que uma vez satisfeitas habilitam a execução de ações associadas. As condições de todas as transições de um módulo podem ser reunidas num único método *selecionar transição para execução*, enquanto que as ações de cada transição devem ser reunidas num método específico. Todos esses métodos pertencem ao tipo de objeto *Corpo_Módulo*.

5.2. Canais de comunicação

O canal não se trata de um conceito arquitetônico propriamente dito, mas sim de um mecanismo que permite especificar as interações independentemente das epecificações dos pontos de interação. A associação de um canal a um ponto de interação impõe restrições a esse ponto, em relação às interações que podem ser emitidas e devem ser recebidas por esse ponto.

A nível de implementação, uma instância de interação é criada no momento da sua emissão ou no momento da sua recepção. A criação de uma instância de interação, no momento da sua recepção, é necessária pois a mesma deve ser codificada e decodificada, sob a forma de parâmetros, para que possa trafegar entre diferentes processos do sistema operacional.

Uma solução possível, para a implementação do mecanismo canal, é a definição de um novo método para criação de instâncias de interação, que capture as restrições estabelecidas pelos canais, aplicando-as no momento oportuno. Esse novo método é dinâmico e por isso deve pertencer a um objeto dinâmico na hierarquia de objetos (vide seção 5.4.).

5.3. Estabelecimento de links de comunicação

Em Estelle o estabelecimento dos links de comunicação é realizado através dos comandos *attach* e *connect*. O primeiro permite a vinculação de pontos de interação entre módulos aninhados, enquanto que o segundo permite a conexão entre pontos de interação ligados por um canal. Um link de comunicação possui dois pontos extremos e zero ou mais pontos intermediários.

O estabelecimento dos links de comunicação, a nível de implementação, é realizado atribuindo-se valores adequados para os atributos da instância do tipo de objeto *Ponto_de_Interação*: *vinculado_para_cima*, *vinculado_para_baixo* e *conectado*. Esses atributos são referências para as outras instâncias do tipo de objeto *ponto_de_interação*, que estão vinculadas ou conectadas à instância em questão.

Um problema que surge é a impossibilidade de se referenciar instâncias pertencentes a outro processo do sistema operacional. Esse problema pode ser superado incluindo-se um novo tipo de objeto à modelagem, que represente o ponto de interação remoto. Esse novo tipo de objeto é chamado *Ponto_de_Interação_Remoto* e deve possuir o atributo *nome_simbólico*, que contém o nome da instância do ponto de interação que está sendo representada. O procedimento para percorrer um link de comunicação é explicado na seção 6.2.

5.4. Cabeçalho e corpo de um módulo

Em Estelle um módulo é especificado em duas partes: cabeçalho e corpo. O cabeçalho define a visibilidade externa do módulo e, consequentemente, o seu tipo. Na especificação de um módulo são definidos o seu identificador, os seus parâmetros e os seus pontos de interação externos, associando-os aos respectivos canais de comunicação. No corpo estão as declarações, as inicializações e as transições do módulo. Uma peculiaridade interessante de Estelle é a possibilidade de se difinir vários corpos para o mesmo cabeçalho. Na criação de uma instância do módulo é associado o seu cabeçalho a um dos corpos, que foram previamente definidos.

A nível de implementação, cada um desses corpos representa um objeto diferente, ainda que eles estejam utilizando o mesmo cabeçalho. Consequentemente, um novo tipo de objeto deve ser acrescentado à modelagem, para diferenciar esses diferentes tipos de módulos. Esse novo tipo de objeto, denominado *Cabeçalho_Módulo_x*, será inserido na hierarquia de tipos como tipo base para o tipo de objeto *Corpo_Módulo*. x é um identificador para os objetos pertencentes a esse novo tipo.

O tipo de objeto *Cabeçalho_Módulo_x* possui como atributos os pontos de interação externos do módulo e como métodos um procedimento para leitura dos parâmetros do módulo, o procedimento já mencionado na seção 5.2 (para a criação das instâncias de interação) e um procedimento para criação das instâncias de pontos de interação.

Uma vez que existirão tantos tipos de objetos quantos forem os possíveis corpos para o *Cabeçalho_Módulo_x*, é necessário identificar cada um desses objetos e, consequentemente, mudar o nome do tipo de objeto *Corpo_Módulo* para *Corpo_y_Módulo_x*. *y* é um identificador para os diversos corpos possiveis e *x* identifica o cabeçalho associado.

5.5. Variáveis exportadas

Em Estelle um módulo filho pode permitir que seu pai tenha acesso (leitura/escrita) as suas variáveis. Essa permissão é concedida na medida em que o módulo filho exporta para o pai essas variáveis. Essa exportação deve ser declarada no cabeçalho do módulo filho.

A nível de implementação esse conceito não encontra suporte diretamente nos recursos da POO, sendo dependente dos recursos da linguagem de implementação utilizada. A implementação desse conceito não é tão simples, pois variáveis globais não podem ser utilizadas, já que os módulos aninhados podem ser processos distintos do sistema operacional. Na seção 6.1 são apresentadas algumas construções que podem auxiliar na resolução desse problema.

6. Atividades de suporte

Cada objeto do tipo *systemprocess* ou *systemactivity* será implementado num processo do sistema operacional. Esse processo será responsável pelo controle do ciclo de execução das transições do módulo que ele representa e das transições dos módulos descendentes que serão, por sua vez, implementados em outros processos ou em procedimentos do processo que representa o módulo pai.

As estruturas de dados e os procedimentos apresentados foram descritos sem considerar a distribuição do sistema. Essa característica dos SDs implica na criação de mecanismos apropriados para a comunicação entre os processos envolvidos.

Outro aspecto, que ainda não foi considerado, é a necessidade de manter o sincronismo entre objetos do tipo módulo, que tenham sido implementados em processos independentes, mas que representam módulos que devem ser executados em paralelo sincronamente. As duas próximas seções apresentam algumas diretrizes para o tratamento dessas questões.

6.1. Comunicação entre os processos

Embora a implementação do mecanismo de comunicação entre processos seja dependente dos recursos disponíveis no ambiente operacional utilizado, algumas diretrizes genéricas podem ser estabelecidas.

As rotinas para tratamento de comunicação constituem um conjunto de procedimentos estáticos, a ser utilizado em qualquer implementação. Portanto, um novo objeto, denominado *Interface_de_Comunicação* e que pertencerá à biblioteca de classes, pode ser acrescentado à modelagem para o tratamento da comunicação. Dessa forma, cada processo do sistema operacional, pertencente ao SD implementado, possuirá uma instância desse novo objeto, cujos métodos oferecem um serviço para a comunicação com os demais processos. Os principais métodos que fazem parte desse objeto são:

- troca de mensagens para a sincronização entre os processos (vide seção 6.2);

- inicialização do processo que contém o objeto que representa um módulo remoto, informando os valores dos atributos correspondentes aos parâmetros desse módulo e correspondente ao nome do módulo pai;

- término da execução do processo que contém o objeto que representa um módulo remoto;

- verificação do recebimento de primitivas de comunicação;

- tranferência de permissão de execução para objetos do tipo módulo, implementados em outros processos;

- envio de requisição de oferta de transição para objetos do tipo módulo, implementados em outros processos.

Para simplificar a implementação desse novo tipo de objeto, pode-se empregar o conceito cliente/servidor. Assim sendo, os métodos do tipo de objeto *Interface_de_Comunicação* fariam chamadas a um servidor de comunicação, que seria um processo independente, que tem por função centralizar o controle das comunicações a nível de sistema operacional. Cada ambiente operacional que contenha um dos processos do SD deve possuir uma cópia desse servidor de comunicação.

Outro conceito, a ser empregado na implementação da comunicação entre os processos, é a estruturação em camadas. Uma primeira camada implementaria algumas funções, dependentes do ambiente operacional, que seriam utilizadas por uma segunda camada mais independente desse ambiente. Essa primeira camada pode ser utilizada tanto na implementação do servidor de comunicação, quanto na implementação dos métodos do tipo de objeto *Interface_de_Comunicação.* Isso facilitaria o trabalho de adaptação do sistema a outras plataformas.

6.2. Sincronismo entre os módulos

Estelle permite a descrição das características de paralelismo e sincronismo na execução dos módulos de uma especificação através dos qualificadores dos módulos. Subsistemas (módulos *systemprocess* ou *systemactivity*) executam em paralelo assincronamente, os módulos filhos de um módulo *systemprocess* ou *process* executam em paralelo

sincronamente e os módulos filhos de um módulo *systemactivity* ou *activity* executam de forma seqüencial.

A nível de implementação, os objetos do tipo *Módulo* devem controlar o disparo dos métodos, que representam as suas transições, e devem controlar a concessão de autorização para os objetos que representam os seus módulos filhos. Essa concessão deve priorizar o disparo dos métodos, observando as restrições de paralelismo e sincronismo definidas pelos qualificadores dos módulos. Esse controle é feito conforme descrito na seção 4.5.

A implementação do ciclo de execução de um módulo poderia ser feita através da utilização de um algorítimo escalonador, centralizando o controle de execução dos métodos relativos às transições. Entretanto, essa solução não é apropriada para implementações distribuídas. O problema pode ser resolvido de forma mais eficiente com a utilização de um protocolo de sincronização, de tal forma que todas as instâncias de objetos do tipo *Módulo* se comportem de acordo com regras locais de sincronização.

A principal vantagem da adoção de um protocolo de sincronização é possibilitar uma distribuição arbitrária das instâncias dos objetos, que representam os módulos de uma especificação, entre os diversos locais de processamento. O protocolo de sincronização, entre duas instâncias de objetos que representam módulos, varia de acordo com os qualificadores desses módulos, pois cada tipo de módulo implementa um ciclo de execução próprio. Na Fig. 9 estão relacionadas as primitivas trocadas por cada tipo de módulo.

Primitivas	1R	1E	2R	2E	3R	3E	4R	4E
(a) Concessão de execução de uma transição		X		X	X	X	X	X
(b) Sinal de fim de execução da transição	X		X		X	X	X	X
(c) Solicitação de oferta de transições disparáveis				X	X	X		
(d) Oferta de transição disparáveis					X	X		
(e) Atualização de variáveis exportadas	X	X	X	X	X	X	X	X
(f) Atualização do contador de tempo		X		X	X	X	X	X

1- systemprocess 3- activity R- Recebe
2- systemactivity 4- process E- Envia

Fig. 9 Primitivas do protocolo de sincronização

As quatro primeiras primitivas são decorrentes dos métodos descritos na Fig.8 (Tabela de Métodos). As primitivas (a) e (b) permitem a um módulo pai, *systemprocess* ou *process*, estabelecer os pontos de sincronismo já mencionados e permitem a um módulo pai, *systemactivity* ou *activity*, estabelecer o sequenciamento da execução de seus descendentes.

A primitiva (e) é um artifício utilizado para a propagação das alterações, ocorridas em

variáveis exportadas, para instâncias de objetos do tipo *Módulo*, que foram implementadas em processos distintos do sistema operacional. Essa primitiva, quando necessária, deve ser emitida ao final de cada ciclo de execução e deve ser recebida pelo destinatário entre o término de um ciclo e o início do próximo ciclo.

A primitiva (f) possibilita a centralização do controle do tempo de execução decorrido, que é utilizado pela cláusula *delay* das transições de Estelle, garantido a sua uniformidade em toda a sub-árvore.

Outro problema que pode ser solucionado, utilizando-se o protocolo de sincronização, é o tráfego das interações através dos links de comunicação. Sua implementação pode ser feita da seguinte maneira: cada subsistema deve possuir uma lista de interações para emissão, que guardará as interações a serem enviadas para outro processo do sistema operacional. Toda vez que um módulo for emitir uma primitiva de comunicação, ele consultará o destino de cada interação que se encontra na lista mencionada. Se o destino de uma interação coincidir com o destino da primitiva a ser enviada, ou coincidir com um nó intermediário do link de comunicação, a interação será codificada sob forma de parâmetros e será anexada à primitiva. O procedimento que implementa o protocolo verificará a existência de interações acopladas à primitiva e, em caso positivo, irá decodificar essas interações e irá armazená-las na *fila_de_interações_recebidas* do ponto de interação apropriado, ou na *lista_de_interações_para_emissão* do nó intermediário, conforme for o caso.

7. Conclusão

O mapeamento dos conceitos arquitetônicos da TDF Estelle nos elementos próprios ao paradigma orientação ao objeto, que foi apresentado neste trabalho, serve de base para o desenvolvimento de ferramentas que visam a implementação de SDs, numa linguagem de programação orientada ao objeto, a partir das especificações formais em Estelle desses SDs.

Tradicionalmente, os compiladores Estelle têm gerado código C ou Pascal. A utilização de uma linguagem de programação orientada ao objeto, como linguagem de implementação, permite reduzir drasticamente a porcentagem de código manual necessária para completar o código gerado pelo compilador. Isso torna o processo de produção da implementação praticamente automático.

Outra grande vantagem da utilização do paradigma orientação ao objeto, é a possibilidade de geração automática de uma implementação distribuída de um SD diretamente a partir da sua especificação. Isto é, se a especificação Estelle do SD em questão contiver diversos subsistemas, não é necessário que a ferramenta trate individualmente cada um desses subsistemas. É possível, a nível de especificação, indicar em qual ambiente operacional cada um dos subsistemas deve ser implantado.

8. Referências

[BOCH 83] BOCHMAM, G.V. Concepts for Distributed Systems Design. Springer-Verlag, 1983, 260 p.

[BOCH 87] BOCHMAM, G.V. Usage of Protocol Development Tools: The Results of a Survey. Protocol Specification, Testing, and Verification, VII, North-Holand, 1987, pp. 139-161

[BOCH 80] BOCHMAM, G.V.; SUNSHINE, C.A. Formal Methods in Communication Protocol Design. IEEE Transactions on Communications, vol.28, pp. 624-631, 1980.

[CORB 90] CORBIN, J.R. The art of distributed applications: programing techniques for remote procedure calls. Sun Microsystems, 1990. 321 p.

[DEMB 89] DEMBINSKI, P.; BUDKOWSKI, S. Specification Language Estelle. North-Holland, 1989, pp. 35-75.

[DOLD 92] DOLDI, L.; GAUTHIER, P. VEDA 2, Power to the protocol designer. Anais da International Conference on Formal Description Techniques 5, Lanion (França), 1992, pp. 37-46.

[EWS 89] ESPIRIT PROJECT 1265 - SEDOS, Bruxelas. EWS User's manual, 1989, 236 p.

[GERB 83] GERBER, G.W. Une Méthode d'Implantation Automatisée de Systèmes Specifiés Formelement. Dissertação de Mestrado, Université de Montréal, Montreal (Canadá), 1983.

[ISO 88] ISO IS 9074. Information Processing Systems - Open System Interconnection - Estelle - A Formal Description Technique Based on an Extended State Transition Model. 1988.

[JARD 86] JARD, C.; GROZ, R.; MONIN, J.F. Experience in Implementing Estelle-X.250 (a CCITT subset of Estelle) in VEDA. Protocol Specification, Testing and Verification, 5, North-Holland, 1986, pp. 315-331.

[LAGE 86] LAGES, N.A.C.; NOGUEIRA, J.M.S. Introdução aos Sistemas Distribuídos. Campinas, Editora da UNICAMP, 1986, 299 p.

[LIN 92] LIN, C.W. Uma Metodologia para Implementação Semi-Automática de Protocolos de Comunicação. Dissertação de Mestrado, Universidade Federal da Paraíba (UFPB), Campina Grande (PB), 1992.

[RUMB 91] RUMBAUGH, J. et alli. Object-Oriented Modeling and Design. Prentice Hall, 1991.

[SIJE 91] SIJELMASSI, R.; STRAUSSER, B. The Distributed Implementation Generator: an

Overview and User Guide. Relatório técnico NCSL/SNA, Gaithersburg (EUA), 1991, 46 p.

[SOUZ 89] SOUZA, W.L. Estelle: Uma Técnica para a Descrição Formal de Serviços e Protocolos de Comunicação. Revista Brasileira de Computação, Rio de Janeiro, vol. 5 (1), jul/set, 1989, pp. 33-44.

[VIEI 93] VIEIRA, M.T.P. Uma Metodologia para o Projeto Lógico de Banco de Dados Orientado para Objetos. XXVI Congresso Nacional de Informática e Telecomunicações, Sucesu 93, Brasília (DF), 1993.