Um implementador semi-automático de protocolos descritos em Estelle para ambiente UNIX

Gilberto Fernandes Taboada[1] **Aloysio de Castro Pinto Pedroza**[1,2]

Universidade Federal do Rio de Janeiro
(1)COPPE - Programa de Engenharia Elétrica
Cx. Postal 68504 CEP 21945-870 - Rio de Janeiro - RJ
Fax : +55 21 290-6626
(2)Departamento de Eletrônica - Escola de Engenharia
Email : gilberto@coe.ufrj.br aloysio@coe.ufrj.br

Resumo

Este artigo faz a descrição de um sistema implementador de protocolos descritos em Estelle para ambiente UNIX. Este implementador faz parte de um sistema maior, composto de um compilador para linguagem Estelle e diversas ferramentas de auxílio ao projetista. O sistema de auxílio e o compilador são brevemente descritos no texto. São feitas algumas propostas de aperfeiçoamento desse implementador, utilizando para isso alguns mecanismos oferecidos pelo sistema operacional UNIX. Com essas alterações propostas, espera-se obter mais versatilidade no uso do implementador e mais desempenho. As estruturas de implementação possíveis graças às alterações são apresentadas em diversas figuras. É também apresentado um exemplo de implementação com a estrutura original do sistema, que mostra que o implementador funciona corretamente.

Abstract

This paper shows the description of an implementation system of protocols specified in Estelle for the UNIX operating system. This implementation system is a part of a major system, composed by a compiler for the Estelle language and many tools that may provide help for a protocol designer. This major system and the compiler are briefly described in the paper. Some proposals of improvement of the implementation system are presented in the text. These proposals use some mechanisms offered by the UNIX operating system. These improvements are expected to allow more versatility in the use of the implementation system, and are also expected to offer more performance of the system. The implementation structures allowed with these improvements are showed in many illustrations. There is also an example of implementation with the original system strucutre that shows that the system works correctly.

I - Introdução

O processo de descentralização de sistemas envolvendo computadores fez surgir a necessidade de padronizar protocolos de comunicação, para que as diversas partes de um sistema pudessem trocar informações entre si. Por isso, é necessário garantir a integridade dos protocolos de comunicação envolvidos para garantir o bom funcionamento de um sistema distribuído.

Há a necessidade de se adotar um método ordenado e claro para especificar esses protocolos. Nesse intuito foram criadas as Técnicas de Descrição Formal (TDF), que permitem que um protocolo seja especificado de forma clara e sem ambigüidades. Entre essas TDF, destaca-se a linguagem Estelle, na qual é baseada a ferramenta descrita neste trabalho.

Essas TDF servem de base para realizar etapas fundamentais para o projeto e desenvolvimento de um protocolo de comunicação, como **simulação**, **verificação** e **testes**. No final, todo o trabalho pode culminar com uma implementação semi-automática do protocolo a partir da própria descrição formal.

O objetivo deste trabalho é apresentar uma proposta de aperfeiçoamento de um implementador semi-automático de protocolos descritos em Estelle, dentro de um projeto maior denominado Sistema de Auxílio ao Desenvolvimento de Protocolos em Estelle.

Este implementador foi desenvolvido em linguagem Modula-2, com várias rotinas em linguagem C, para estações de trabalho SunSparc com sistema operacional SunOS v. 4.1.2.

O aperfeiçoamento desse implementador inclui alterações visando melhorias no funcionamento do motor da especificação, especificação de uma interface de comunicação entre o código gerado e as aplicações do usuário e especificação de uma interface de acesso a uma placa de rede padrão Ethernet aproveitando-se dos recursos da plataforma descrita acima.

Na seção II são descritos alguns aspectos mais importantes da linguagem Estelle. Também é realizada uma breve descrição do Sistema de Auxílio ao Desenvolvimento de Protocolos em Estelle e descrição do compilador Estelle desse sistema. Na seção III faz-se uma breve discussão sobre alguns conceitos de implementação semi-automática de protocolos e é descrito o implementador e sua estrutura original. Na seção IV são descritas as alterações propostas no implementador, as melhorias propostas no motor da especificação e as características das interfaces propostas com o usuário e com o dispositivo de rede. Na seção V são apresentados os resultados de um exemplo de implementação, e finalmente na seção VI são apresentadas as conclusões finais.

II. - O Sistema de Auxílio ao Desenvolvimento de Protocolos em Estelle

II.1 - A linguagem Estelle

Estelle é uma linguagem para especificação de sistemas distribuídos, voltada principalmente para especificação de protocolos de comunicação.

Uma especificação em Estelle é uma descrição de um conjunto de máquinas de estado que se comunicam umas com as outras, e cujas ações internas são descritas por comandos da linguagem Pascal.

Principais características da linguagem Estelle :

Instâncias de Módulos

Cada máquina de estado de uma especificação é conhecida por instância de módulo, ou tarefa. Uma instância de módulo é dita “ativa” se em seu interior estiver definida pelo menos uma transição. Cada instância de módulo pode ter os seguintes atributos :

- systemprocess
- systemactivity
- process
- activity

O atributo de uma instância de módulo é fundamental para determinar como será o seu funcionamento interno e o de suas instâncias de módulo filhas.

As instâncias de módulo podem ser definidas umas dentro das outras, numa estruturação semelhante à da linguagem Pascal (referente a *functions* e *procedures* umas definidas no interior de outras). Há no entanto uma série de regras regendo essa estruturação.

Na hora da escolha de uma transição a ser disparada, deve se respeitar a prioridade pai-filho : uma transição pronta para ser disparada de uma instância de módulo qualquer tem a preferência sobre as transições das suas instâncias de módulo descendentes.

Nas instâncias de módulo com atributo *systemprocess*, a cada passo computacional, todas as transições prontas para ser disparadas que não estiverem em conflito pai-filho são selecionadas para serem disparadas simultaneamente.

Nas instâncias de módulo com atributo *systemactivity*, a cada passo computacional, somente uma das transições prontas para serem disparadas e que não estiverem em conflito pai-filho é selecionada para ser disparada. Essa escolha é feita aleatoriamente.

Comunicação entre instâncias de módulo

Há dois mecanismos de comunicação entre instâncias de módulo em Estelle : compartilhamento de variáveis e troca de mensagens.

Variáveis podem ser compartilhadas entre uma instância de módulo e sua instância de módulo pai. Essas variáveis são declaradas como *exported* pela instância de módulo filha. Não há problema de haver acesso simultâneo a essas variáveis devido a regra da prioridade pai-filho: ações da instância de módulo pai sempre têm prioridade.

Cada instância de módulo pode ter pontos de entrada e saída chamados pontos de interação. Esse pontos de interação podem ser internos ou externos. As instâncias de módulo podem trocar mensagens entre si através desses pontos de interação. Uma instância de módulo envia uma mensagem (ou interação) através de um canal formado pela ligação de 2 pontos de interação de instâncias de módulo diferentes.

A interação é colocada no final de uma fila tipo FIFO (First In First Out) ilimitada, associada ao ponto de interação da instância de módulo que está recebendo a mensagem.

Uma interação pode sempre enviar uma interação, não há necessidade de haver um "rendez-vous" com a instância de módulo que está recebendo a mensagem. Existe entretanto o mecanismo de "rendez-vous" numa extensão da linguagem chamada Estelle*, apresentada em [5].

Módulos

Um módulo em Estelle é um par composto de cabeçalho e corpo de módulo. O cabeçalho define um tipo de módulo, cujos valores são módulos com a mesma visibilidade externa, ou seja : mesmos pontos de interação, mesmas variáveis exportadas e mesmo atributo de classe.

O corpo contém a definição do comportamento interno do módulo. Para cada cabeçalho deve ser definido pelo menos um corpo. O corpo é composto basicamente de 3 partes : declaração, inicialização e transição.

A parte de declaração contém declarações típicas da linguagem Pascal (constantes, variáveis, procedures) e declarações de objetos de Estelle, tais como módulos, variáveis módulos (que servem de referência para instâncias de um certo tipo de módulo), estados, conjuntos de estados e pontos de interação internos (para comunicação com módulos filhos).

Na parte de inicialização diversas variáveis têm seus valores especificados. Assim toda instância desse módulo que for criada terá essas variáveis com os mesmos valores. São inicializadas as variáveis Pascal, o estado do módulo e as variáveis módulo internas a esse módulo.

A parte de transição contém a descrição do comportamento interno do módulo. Toda declaração de transição é composta de 2 partes : **condição** da transição e **ação**.

Cláusulas de **condição** : *from, when, provided, priority, delay*.

Essas cláusulas impõem uma série de condições para que a transição em questão possa ser considerada apta a ser disparada.

Cláusulas de **ação** : *to,* bloco de transição.

"to" especifica para qual estado a máquina passa depois do disparo da transição. No bloco de ação estão especificados os comandos que a máquina deve obedecer durante aquela transição.

Primitivas de Estelle

São os comandos da linguagem Estelle que manipulam os objetos típicos da linguagem, como pontos de interação e instâncias de módulos, como nos exemplos a seguir:

release X - destrói a instância de módulo referenciada mela variável módulo *X*.

disconnect p - desconecta o ponto de interação *p* do ponto ao qual estava conectado.

disconnect X - desconecta todos os pontos de interação da instância de módulo *X*.

output p1.m - envia a interação *m* pelo ponto de interação *p1*. Se o ponto de interação *p1* estiver conectado ao ponto de interação *p2*, isso faz com que a mensagem *m* seja depositada no final da fila de mensagens associada ao ponto *p2*.

II.2 - O sistema de auxílio ao projeto de protocolos de comunicação em Estelle

O sistema de auxílio ao projeto de protocolos em Estelle tem como objetivo oferecer ao usuário projetista ferramentas que facilitem o desenvolvimento de protocolos de comunicação. O sistema é apresentado em [4].

Esse sistema é composto de vários módulos, como pode-se ver na figura 1, a saber :

- Editor dedicado
- Compilador
- Simulador
- Verificador
- Gerador de seqüências de teste
- Implementador (em *hardware* e *software*)
- Interface gráfica

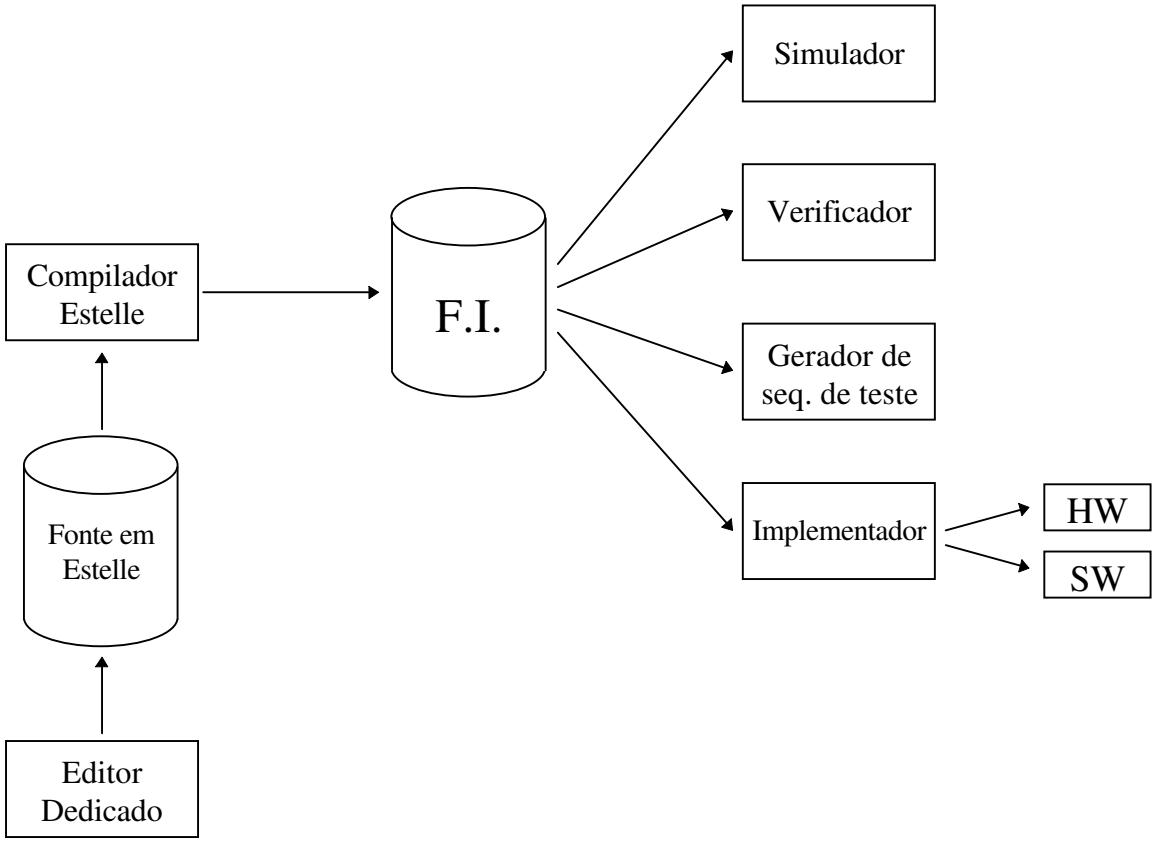


fig. 1 - O Sistema de Auxílio em detalhe

O usuário, a partir da interface gráfica, tem acesso a todos os componentes do sistema. Com o auxílio de um editor dedicado, faz-se a especificação do protocolo em linguagem Estelle. Com a compilação do arquivo da especificação, é gerada uma saída, chamada Forma Intermediária. Essa Forma Intermediária é que vai servir de base para as demais ferramentas do sistema (simulador, verificador, gerador de seqüências de teste e implementador, que pode realizar uma implementação a nível de *software* ou *hardware*). A seguir, são apresentados alguns detalhes do compilador, e em particular detalhes das suas saídas, os arquivos da Forma Intermediária.

II.2 - O Compilador Estelle

O compilador faz a tradução de um arquivo-fonte em Estelle para uma Forma Intermediária, a partir da qual todas as outras ferramentas do sistema vão poder trabalhar. Este compilador é apresentado em [6]

O funcionamento do compilador é feito em três etapas : análise léxica, análise sintática e análise semântica, esta última dividida em análise de contexto e geração de código. A geração de código é na realidade a geração da Forma Intermediária, como pode-se ver na figura 2.

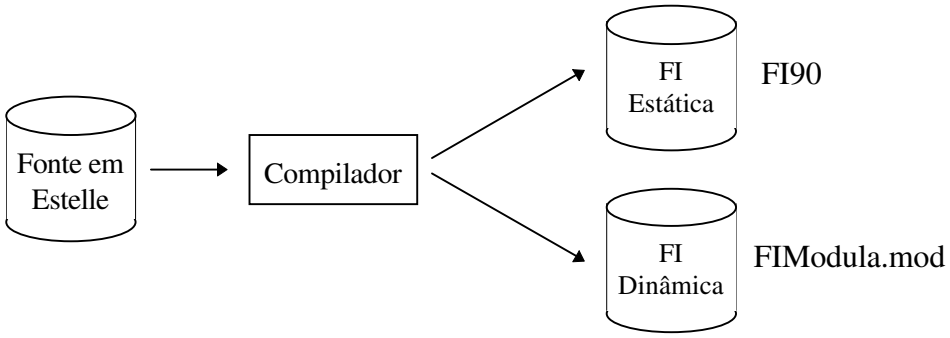


fig. 2 - Os arquivos de saída do compilador

A Forma Intermediária consiste em 2 arquivos nos quais está descrita a estrutura da especificação numa forma que pode ser utilizada pelos demais módulos do Sistema de Auxílio. Esses dois arquivos são os seguintes: FI90 (correspondente à Forma Intermediária Estática) e FIModula.mod (correspondente à Forma Intermediária Dinâmica).

Forma Intermediária Estática

É um arquivo texto, contendo diversas listas onde estão as descritas as estruturas dos objetos de Estelle da especificação. Esses objetos são, tipicamente: cabeçalhos de módulos, corpos de módulos, canais, interações, pontos de interação, nomes de transições e cláusulas de transições.

Forma Intermediária Dinâmica

É um arquivo texto em linguagem Modula-2 que contém a informação relativa ao comportamento da especificação em tempo de execução, ou seja: as cláusulas “provided” das transições, as ações da transições e as estruturas típicas da linguagem Pascal que se encontram no interior da especificação (constantes, tipos, variáveis, funções e procedimentos).

III - O Implementador de Protocolos Descritos em Estelle

III.1 - Implementação semi-automática de protocolos

Implementação semi-automática de um protocolo é o processo pelo qual, a partir de uma especificação formal do protocolo, gera-se de forma semi-automática um código que implemente suas funções.

O código que compõe uma implementação semi-automática pode ser classificado em dois tipos: a parte independente do ambiente para o qual se está gerando e código, e a parte dependente do ambiente. A parte independente é aquela que descreve as ações das máquinas de estado nas transições. Esta é a parte do código que é implementada de forma semi-automática, e que, dependendo do caso, chega a constituir de 50% a 75% do código total da implementação. A parte dependente do ambiente diz respeito a mecanismos como: detecção de eventos, gerenciamento de memória e comunicação entre diferentes camadas de protocolos. Em Estelle, pode-se citar os exemplos das cláusulas *provided*, *when*, *delay*, e ainda os mecanismos de comunicação entre módulos (tanto as variáveis compartilhadas como as trocas de mensagens). Esta parte do código tem que ser feita a mão pelo usuário projetista, pois em cada ambiente (conjunto máquina + sistema operacional) há uma maneira particular de implementar esses mecanismos. Entretanto, uma vez criado esse código, ele pode ser usado para novas implementações de diferentes protocolos, normalmente com poucas ou até mesmo nenhuma modificação.

Há dificuldades entretanto que devem ser superadas, principalmente no que diz respeito à compatibilidade entre os tipos definidos em Estelle e os tipos disponíveis na linguagem de implementação. Entretanto, as vantagens proporcionadas por esse tipo de implementação compensam essas dificuldades.

É fácil mostrar que uma implementação gerada de forma semi-automática está em acordo com a norma especificada - a partir do instante que foram feitas simulações com o protocolo e ele se comportou de forma correta, se o processo de implementação estiver correto, o código gerado vai se comportar como esperado.

O código gerado a partir da implementação semi-automática é fácil de ser mantido. Caso se queira modificar algo no protocolo (para testes ou no caso de as normas do protocolo serem modificadas, por exemplo), basta fazer a modificação na especificação formal e gerar um novo código de implementação.

O processo de implementação semi-automática pode proporcionar uma considerável redução no tempo de desenvolvimento de um protocolo. Isso porque o processo gera de forma semi-automática de 50% a 75% do código total. Além disso, esse código não está sujeito a falhas típicas de um projetista humano, fazendo assim mais fácil o trabalho de depuração.

Por fim, um processo de implementação semi-automática pode gerar um código tão eficiente (tanto em tamanho quanto em velocidade) quanto um código feito inteiramente de forma manual. Isso só depende da eficiência do sistema de implementação propriamente dito.

III.2 - O IPDE - Implementador de Protocolos Descritos em Estelle

O sistema do implementador (IPDE), assim como todas as outras ferramentas do Sistema de Auxílio descrito anteriormente, utiliza como entrada os arquivos gerados pelo compilador Estelle - os arquivos FI90 (da Forma Intermediária Estática) e o arquivo FIModula.mod (Forma Intermediária Dinâmica). O implementador na sua forma original é apresentado em [8].

O funcionamento do processo de implementação funciona da seguinte maneira: após a compilação da especificação Estelle (processo que equivale à geração do código independente do ambiente), faz-se a compilação do arquivo FIModula.mod (com um compilador para linguagem Modula-2), e em seguida o arquivo-objeto resultante é "linkado" com o restante do código do implementador (equivalente ao código feito à mão pelo projetista). A partir daí, tem-se um código executável que implementa as funções do protocolo especificado.

A fase de funcionamento do código executável é dividida em três etapas distintas: leitura da Forma Intermediária Estática (e construção das estruturas de dados correspondentes), inicialização dos módulos da especificação e início do funcionamento do protocolo, conhecido também por Motor da Especificação.

Leitura da FI Estática

Como já foi dito anteriormente,. a FI Estática está toda descrita num arquivo-texto. Na primeira etapa do funcionamento, o IPDE vai ler o arquivo da FI Estática e armazenar na memória as estruturas de dados equivalentes, cada dado no interior do objeto correspondente. Esses objetos são descritos a seguir. Todos eles são apresentados em [6] e [7].

•Objeto Sb - contém a identificação de todos os corpos de módulo existentes na especificação.

•Objeto BodySymbol - descreve a hierarquia dos corpos de módulo contidos no objeto Sb (quais módulos foram definidos no interior de quais módulos).

•Objetos Sinh_id, Ssyph_id, Ssyah_id, Sph_id, Sah_id - contém as informações relativas aos nomes e tipos de cabeçalhos de módulo (inativo, *systemprocess*, *systemactivity*, *process* ou *activity*).

•Objetos Sinit_id, Sinput_id, Sspont_id, Sot - contêm respectivamente as informações sobre as transições de inicialização, transições de entrada (*input transitions*, com a cláusula "when"), transições espontâneas (sem a cláusula "when") e de todas as transições da especificação.

•Objeto Sms_id - contém os nomes dos estados dos módulos da especificação.

•Objetos Siip_id, Seip_id, Siqip_id, Scqip_id, Snqip_id - contêm respectivamente as informações referentes aos pontos de interação internos, pontos de interação externos, pontos de interação com fila individual associada, com fila comum associada, e aqueles que utilizam o mecanismo de "rendez-vous" para troca de mensagens.

•Objeto Sint_id - contém os nomes das interações definidas nos canais contidos em cada corpo de módulo

•Objeto Sm - contém os nomes das variáveis módulo existentes para cada tipo de cabeçalho de módulo.

•Objetos Sdmdim e Sdipdim - contêm respectivamente as informações relativas às dimensões dos *arrays* de variáveis módulo e *arrays* de pontos de interação da especificação.

•Objeto Transitions - associa a cada transição da especificação o valor da cláusula "delay" associada (caso não exista essa cláusula, associa valor 0) e a sua prioridade.

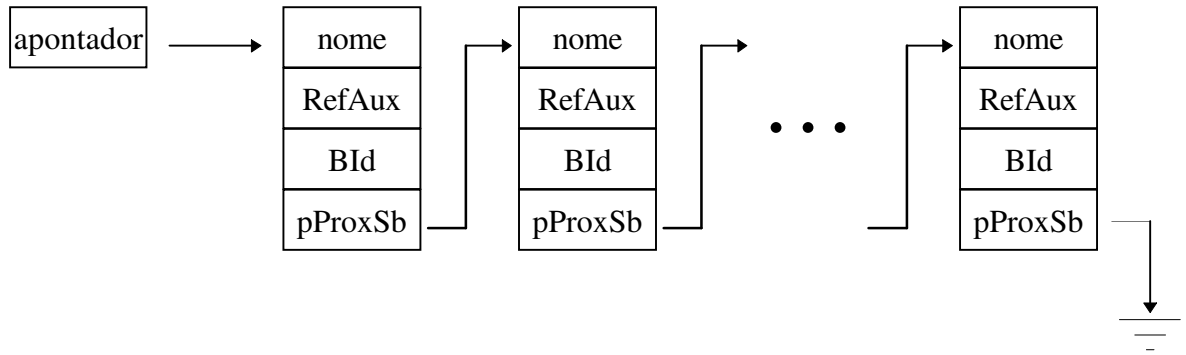
•Objeto IpTrans - contém informações complementares relativas ao tipo de mecanismo de comunicação utilizado em cada transição.

•Objetos Sinput_State e Soutput_State - associam a cada transição o estado de entrada, ou seja, a partir de qual estado aquela transição pode ser disparada (equivale à cláusula “from” de Estelle) e o estado de saída, para o qual a máquina passa depois do disparo da transição (equivale à cláusula “to” de Estelle).

•Objeto Role - associa a cada ponto de interação as interações que podem ser enviadas e recebidas por ele.

Todos esses objetos armazenam os dados correspondentes em filas encadeadas. Em alguns casos, pode haver várias listas encadeadas, umas apontando para outras. Abaixo são mostrados dois exemplos, nas figuras 3 e 4.

Objeto Sb



Nome : cadeia de caracteres contendo o nome do corpo de módulo

RefAux : número inteiro utilizado para funções internas do implementador

BId : número inteiro utilizado para identificação do corpo de módulo

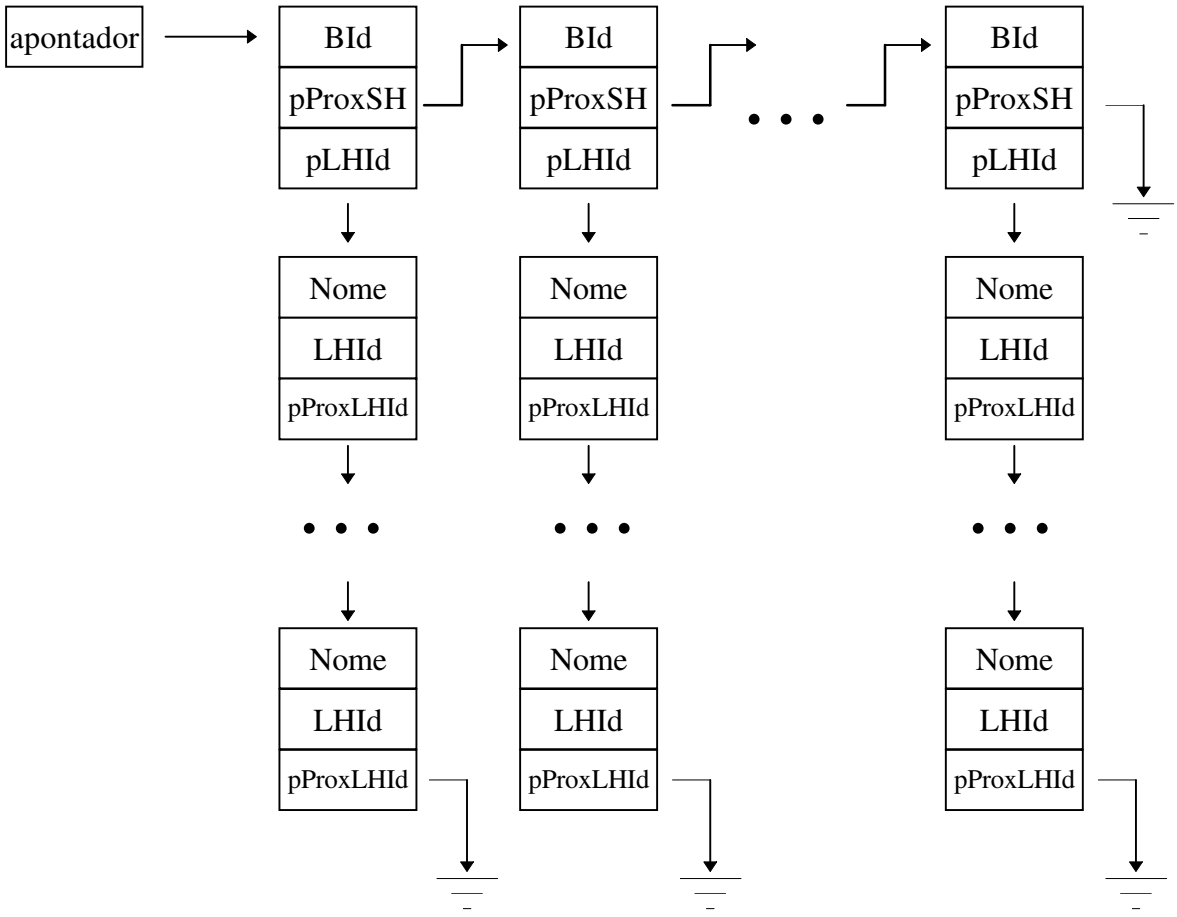
pProxSb : ponteiro para o próximo corpo de módulo da lista

fig. 3 - Estrutura de dados do objeto Sb

Alguns dos objetos de Estelle podem ser armazenados em filas simples, como essa mostrada acima. Entretanto, alguns deles devem ser armazenados em conjuntos de filas por ser necessário manter dois tipos de informação sobre eles: a qual corpo de módulo pertencem e a identificação do objeto no interior do módulo. Um exemplo é o objeto Sah_id, apresentado na figura 4.

Todos esses dados são encapsulados no interior dos respectivos objetos. O usuário não tem acesso direto a eles, mas dispõe de funções que podem manipular esses dados. Assim, pode-se copiar os dados para outra região de memória e manipulá-los à vontade, mas os dados originais permanecem inalterados.

Objeto Sah_id



BId: número inteiro identificador do corpo de módulo

pProxSH: apontador para o próximo corpo de módulo

pLHId: apontador para a lista de cabeçalhos de módulo definidos no interior do corpo de módulo atual

Nome: cadeia de caracteres contendo o nome do cabeçalho de módulo

LHId: número inteiro que identifica o cabeçalho de módulo

pProxLHId: apontador para o próximo cabeçalho de módulo

fig. 4 - Estrutura de dados do objeto Sah_id

Após a leitura da FI Estática, começa o processo de inicialização da especificação. Começa aí a notar-se uma das características mais importantes do arquivo FIModula.mod, gerado pela compilação da especificação em Estelle. Neste arquivo, são fornecidas ao usuário as funções que permitem manipular as variáveis Pascal da especificação, e são fornecidas também as próprias instruções que devem ser executadas no disparo de uma transição. Entretanto, supõe-se que o projetista vai fornecer certas estruturas de dados e certas funções a este arquivo. Entre essas funções encontram-se as próprias primitivas de Estelle. São dados e funções que normalmente devem ser implementados de forma diferente em cada ambiente, e que portanto devem ficar a cargo do projetista. As primitivas encontram-se todas definidas no arquivo Sprimitives.mod, e a maioria das estruturas de dados encontram-se no arquivo Stype.def. Também são importados pelo arquivo da FI Dinâmica todas as funções declaradas como *primitive* ou *external* na especificação, e que estão definidas no arquivo declarado com a diretiva de compilação {$ external }. No arquivo da FI Dinâmica, entre outras funções, são exportadas as funções *predicado* e *ação*. A primeira testa se o predicado de uma transição tem valor verdadeiro (*TRUE*). A segunda executa os comandos da transição passada como parâmetro.

Assim, durante essa fase é inicializado o módulo mais externo da especificação, e são então executados os comandos definidos nessa transição, tais como: instanciação de módulos-filhos, inicialização de variáveis Pascal e ligações entre pontos de interação (*connect* e *attach)*. São inicializados também vários dados de interesse, tais como as variáveis que indicam o estado de cada instância de módulo da especificação.

Após essa fase de inicialização, dá-se o início do funcionamento do "motor" da especificação, a rotina que vai fazer as máquinas de estado da especificação funcionarem de fato. Esse motor funciona num *loop* infinito, que basicamente realiza os seguintes passos:

1. Atualização do sistema - neste passo é atualizado o relógio interno da especificação, assim como o estado da instância de módulo que teve uma transição disparada no último passo computacional, ou as instâncias, no caso de terem sido disparadas várias transições em paralelo.
2. Seleção das transições que podem ser disparadas - todas as transições oferecidas para serem disparadas têm as suas condições testadas, utilizando-se para isso a função *predicado* exportada pelo arquivo da FI Dinâmica.
3. Escolha de uma transição - uma transição é escolhida para ser disparada entre aquelas que foram selecionadas no passo anterior. Essa escolha obedece os critérios definidos na linguagem Estelle, como prioridade pai-filho e escolha segundo os atributos das instâncias de módulo.
4. Execução da transição - é realizado o disparo da transição selecionada, e assim realizados os comandos definidos para a transição no interior da especificação Estelle. Para isso, utiliza-se a função *ação* exportada pelo arquivo FIModula.mod.
5. Retorno ao passo 1.

Para o teste da cláusula “when”, o motor verifica se há uma mensagem depositada na fila associada ao ponto de interação em questão, e verifica de que tipo é essa mensagem. Para a verificação da cláusula “delay”, faz-se a comparação do tempo decorrido desde o instante em que o predicado da transição foi considerado verdadeiro. A função *predicado* verifica a cláusula *provided* da transição. No caso de não haver nenhuma transição para ser disparada, o ciclo continua se repetindo até que haja alguma transição para ser disparada - isto caracteriza um caso de espera ocupada, que consome bastante tempo de CPU.

IV - Propostas de melhorias no IPDE

O sistema do implementador pode ser melhorado em alguns aspectos, para torná-lo mais flexível e mais eficiente. Essas melhorias podem ser divididas em dois tipos: desenvolvimento de uma interface de comunicação com aplicações do usuário e melhorias no motor da implementação. Nos dois casos, utiliza-se recursos disponíveis no sistema operacional UNIX, para o qual foi desenvolvido o implementador.

IV.1 - Interface de comunicação

A versão atual do implementador, tal como foi apresentada, dispõe somente de um tipo de interface externa - uma interface que permite à especificação comunicar-se com a camada de rede. Foi desenvolvido um módulo que utiliza funções externas que permitem o uso das primitivas do protocolo TCP/IP. Dessa maneira, qualquer módulo desenvolvido pelo usuário transmite mensagens para esse módulo, que em seguida o transmite à rede. Não há qualquer tipo de interface para aplicações do usuário. Desta forma, a própria aplicação deve ser desenvolvida no interior da especificação Estelle.

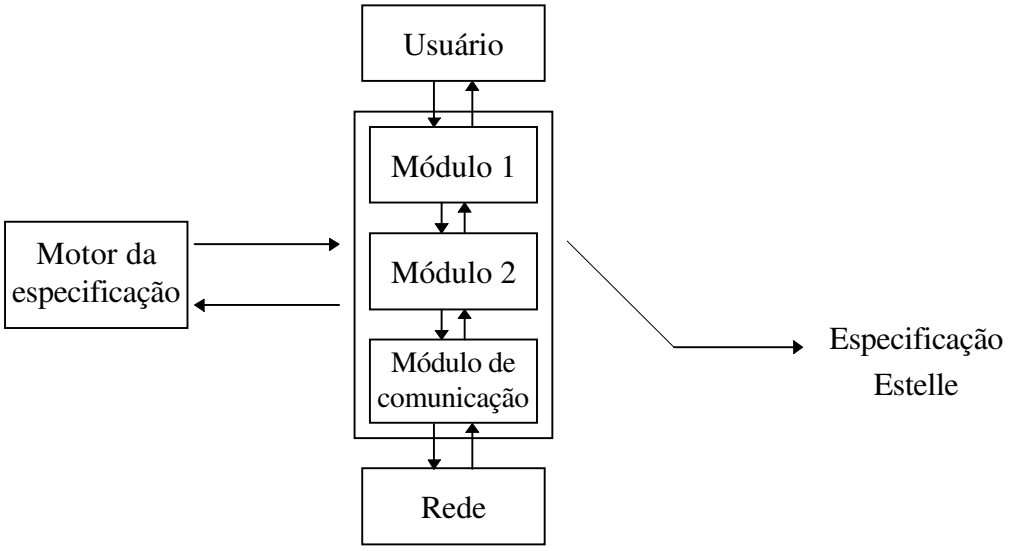


fig. 5 - esquema de funcionamento de uma implementação

A interface proposta inclui o uso do mecanismo de *pipe* disponível no sistema operacional UNIX. Um *pipe* é na verdade um par de "tubos" que permitem que dois processos troquem mensagens. Assim, um módulo Estelle seria o responsável pela manipulação desse *pipe*, e passaria para outros módulos as mensagens que chegassem ao seu interior, seja pela interface de rede, seja por um processo do usuário. Essa nova interface permite também que o usuário desenvolva a sua aplicação em separado da especificação Estelle, na linguagem que mais lhe convier. Outra possibilidade é a separação dos módulos de Estelle em diferentes processos. Na versão atual, uma especificação Estelle é tratado como um único processo pelo sistema operacional. Com a nova interface, torna-se possível para o usuário separar os módulos em diferentes processos, e a comunicação entre eles, ao invés de ficar a cargo da especificação, fica a cargo dos módulos de comunicação. Por fim, essa interface pode servir tanto para comunicação com o usuário quanto para com a interface de rede, seja ela qual for, a nível de rede (TCP/IP) ou mesmo a nível de acesso ao meio (IEEE 802.3).

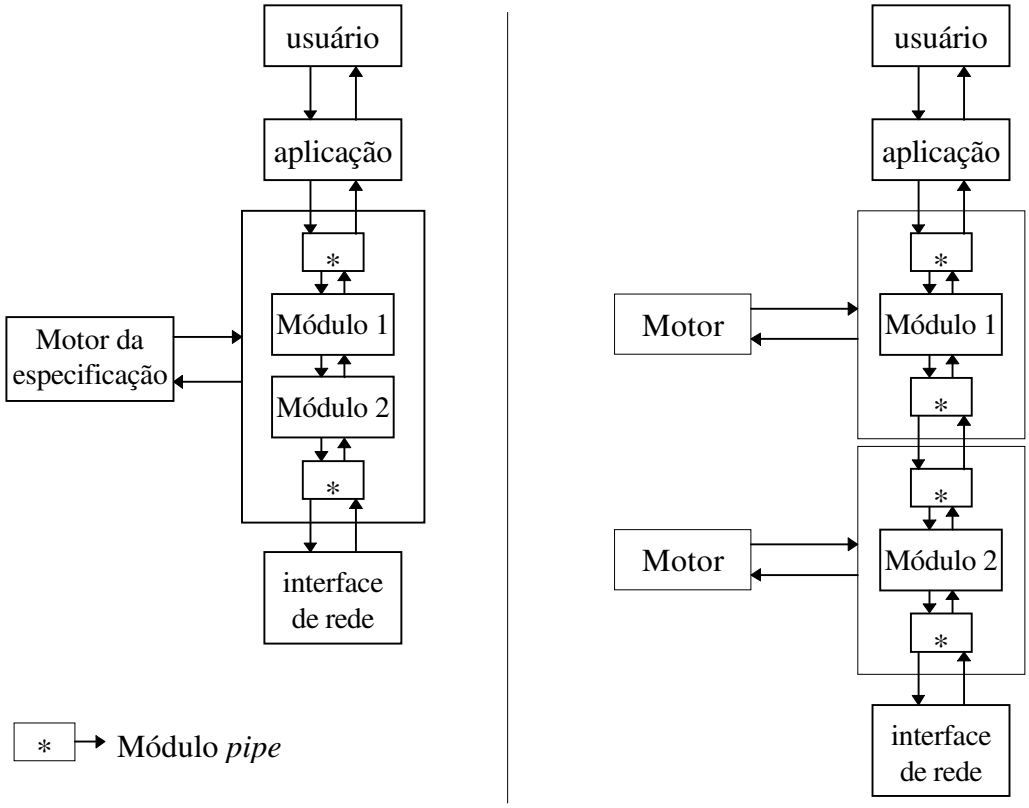


fig. 6 - exemplos de possibilidades de implementação

IV.2 - Alterações no motor

A proposta de melhoria no motor envolve acabar com o mecanismo de espera ocupada, substituindo-o por um mecanismo de detecção de eventos que não gaste tempo de CPU com varredura. Para tal fim, pode-se usar a chamada ao sistema *select*, cuja função é esperar permissão de leitura e/ou escrita em um ou mais descritores de arquivo, sem gastar tempo de CPU.

Assim, a cada ponto de interação associa-se um descritor de arquivo. Quando uma mensagem é depositada na fila associada ao ponto de interação, uma mensagem é enviada ao descritor de arquivo correspondente. Dessa forma, o motor, ao invés de ter que verificar cada fila associada a cada ponto de interação, somente precisará esperar por mensagens nos descritores de arquivos correspondentes. Quando aparecer alguma mensagem num desses descritores, o motor captura a mensagem disponível no topo da fila associada ao ponto de interação correspondente.

A chamada *select* também oferece um mecanismo de temporização, que permite que o usuário retome o controle do programa se depois de um determinado tempo nenhuma mensagem chegar a nenhum dos descritores de arquivos controlados pela chamada. Esse mecanismo pode ser utilizado para disparar as transições espontâneas com cláusula “delay”. Dessa forma, todas as transições podem ser tratadas por essa chamada.

A expectativa é que se obtenha uma ligeira melhora no desempenho do implementador, já que dessa forma elimina-se a espera ocupada, deixando livre mais tempo de CPU para outros processos do usuário que estejam rodando em paralelo com o código gerado pelo implementador, como por exemplo a aplicação que estiver utilizando o protocolo gerado por esse método.

V - Exemplo de implementação

Para efeito de demonstração, foi criada uma especificação simples, com um exemplo de um sistema produtor-consumidor. Nesta especificação estão descritas duas máquinas de estado: o produtor e o consumidor. O produtor é inicializado e fica esperando que o usuário aperte uma tecla para começar a produzir mensagens e enviá-las ao consumidor. Este último é inicializado e fica esperando as mensagens, e quando as recebe ele exibe na tela o número de seqüência da mensagem. A listagem da especificação encontra-se no apêndice A.

Em Estelle, pode-se ilustrar o sistema especificado como na figura 7 abaixo:

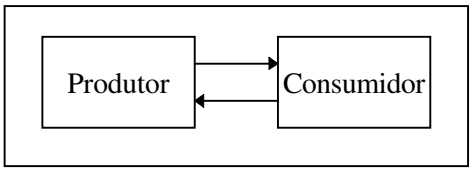


fig. 7 - Sistema Produtor-Consumidor

A saída gerada encontra-se no apêndice B, e por ela pode-se perceber que o sistema implementado funcionou de forma correta. Assim, pode-se supor que qualquer protocolo que tenha sido corretamente especificado deverá funcionar como esperado.

VI - Conclusões

Neste trabalho foi feita a descrição de um implementador semi-automático de protocolos descritos em Estelle. O implementador, como foi visto, gera o código executável de forma correta. Foram também feitas propostas de aperfeiçoamento neste implementador, principalmente no que diz respeito à interface com o usuário e com a rede, e melhorias no motor da implementação. Com essas alterações, aproveitando-se dos mecanismos oferecidos pelo sistema operacional UNIX, espera-se que o implementador tenha mais flexibilidade em seu uso, e espera-se também que haja melhoras no desempenho com a eliminação da espera ocupada. As alterações propostas encontram-se atualmente em estágio avançado de implementação.

VII - Referências

[1] S. Budkowski, P. Dembinski, "An Introduction to Estelle: A Specification Language for Distributed Systems", *Computer Networks and ISDN Systems*, vol. 14, pp. 3-23, Elsevier Science Publishers B.V. (North Holland), 1987.

[2] P. Dembinski, S. Budkowski, "Specification Language Estelle", *The Formal Description Technique Estelle*, pp 35-75, Elsevier Science Publishers B.V. (North Holland), 1989.

[3] D. P. Sidhu, T. P. Blumer, "Semi-automatic Implementation of OSI Protocols", *Computer Networks and ISDN Systems*, vol. 18, pp. 221-238, Elsevier Science Publishers B.V. (North Holland), 1990.

[4] A.C.P. Pedroza, P.R.O. Valim, C.C. Goulart, R.C. Oliveira Jr., "Um sistema de auxílio ao projeto de protocolos de comunicação para redes de computadores", *Anais do Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos*, pp. 80-87, setembro de 1989.

[5] J.P. Courtiat, "Estelle*: a powerful dialect of Estelle for OSI protocol description", *Protocol Specification, Testing and Verification VIII*, pp. 171-186, Elsevier Science Publishers B.V. (North Holland), 1988.

[6] R.C. de Oliveira Jr., "Um compilador para a linguagem Estelle", Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEL/COPPE/UFRJ, 135 p., Rio de Janeiro, agosto de 1991.

[7] P.R.O. Valim, "Um simulador de protocolos de comunicação para a linguagem Estelle", Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEL/COPPE/UFRJ, 130 p., Rio de Janeiro, abril de 1990.

[8] S.C. de Alcântara, "Implementação semi-automática de protocolos descritos em Estelle", Tese de Mestrado, Universidade Federal do Rio de Janeiro, PEL/COPPE/UFRJ, Rio de Janeiro, março de 1993.

[9] N. Wirth, "Programando em Modula-2", LTC - Livros Técnicos e Científicos Ed., 1989.

[10] Sun Microsystems, "Users Reference Manual", Revision A, março de 1990.

Apêndice A

Listagem da especificação ProdCons.est

```
Specification ProdutorConsumidor systemactivity;
  default individual queue;
  timescale seconds;
  {$ external NetUserUDP }

  type
    string   = array [0..255] of char;
    MSG_Type = record
                 tipo  : integer ;
                 dados : array [0..15] of char ;
               end ;

  procedure ClrScr;                          primitive;
  procedure WriteLn;                         primitive;
  procedure WriteInt    ( i     : integer ); primitive;
  procedure WriteString ( s     : string );  primitive;
  function  KeyPressed  : BOOLEAN;           primitive;

  Channel INTERFACE_type (Lado_1, Lado_2);
    by Lado_1 : MS_1 (Mensagem : MSG_Type);
    by Lado_2 : MS_2 (Mensagem : MSG_Type);

  Module PRODUTOR_type activity;
    ip
      P : INTERFACE_type (Lado_1) ;
  end ; (* PRODUTOR_type *)

  Body PRODUTOR_body for PRODUTOR_type ;
    var
     counter : integer ;
      msg     : MSG_Type ;

    state
      ZERO, E1, E2 ;

    initialize  to ZERO
    name INIT_PRODUTOR :
    begin
      counter := 0 ;
      WriteString ('Modulo PRODUTOR inicializado, esperando tecla...') ;
      WriteLn ;
    end ; (* initialize *)
```

```
  trans
    from ZERO to E1
    provided KeyPressed
    name PRODUZ_PRIMEIRO :
    begin
      counter  := counter + 1 ;
      msg.tipo := counter ;
      WriteString ('Produtor enviando mensagem ') ;
      WriteInt (counter) ;
      WriteLn ;
      output P.MS_1 (msg) ;
    end ;

  trans
    from E1 to E1
    provided (counter <= 10)
    delay (2)
    name PRODUZ_E_ENVIA :
    begin
      counter  := counter + 1 ;
      msg.tipo := counter ;
      WriteString ('Produtor enviando mensagem ') ;
      WriteInt (counter) ;
      WriteLn ;
      output P.MS_1 (msg) ;
    end ;

  trans
    from E1 to E2
    provided (counter > 10)
    name TERMINA_PRODUTOR :
    begin
      WriteString ('Produtor Terminou') ;
      WriteLn ;
    end ;
end; (* PRODUTOR_body *)

Module CONSUMIDOR_type activity ;
  ip
    C : INTERFACE_type(Lado_2) ;
end; (* CONSUMIDOR_type *)

Body CONSUMIDOR_body for CONSUMIDOR_type;

  state
    C0, C1, C2 ;

  initialize to C0
```

```
    name INIT_CONSUMIDOR :
    begin
      WriteString ('Consumidor inicializado... ') ;
      WriteLn ;
    end ; (* initialize *)

    trans
      from C0 to C1
      when C.MS_1
      name RECEBENDO_PRIMEIRA :
      begin
        WriteString ('Recebi mensagem ') ;
        WriteInt (Mensagem.tipo) ;
        WriteLn ;
      end ;

    trans
      from C1 to C2
      name T_VAZIA :
      begin
      end ;

    trans
      from C2 to C1
      when C.MS_1
      name RECEBENDO_MENSAGEM :
      begin
        WriteString ('Recebi mensagem ') ;
        WriteInt (Mensagem.tipo) ;
        WriteLn ;
      end ;

  end (* CONSUMIDOR_body *);

  modvar
       prod  : PRODUTOR_type ;
       cons  : CONSUMIDOR_type ;

  initialize
    begin
      ClrScr ;
      init  prod  with  PRODUTOR_body ;
      init  cons  with  CONSUMIDOR_body ;
      connect  prod.P to cons.C ;
      WriteString ('Conectou pontos de interacao (ProdCons.est) ');
      WriteLn ;
    end ; (* initialize *)
end (* Pai_Filho *).
```

Apêndice B

Saída gerada pela implementação da especificação ProdCons.est

```
Conectou pontos de interacao (ProdCons.est)
Modulo PRODUTOR inicializado, esperando tecla...
Consumidor inicializado...

 Inicio da Execucao ...

Produtor enviando mensagem 1
Recebi mensagem 1
Produtor enviando mensagem 2
Recebi mensagem 2
Produtor enviando mensagem 3
Recebi mensagem 3
Produtor enviando mensagem 4
Recebi mensagem 4
Produtor enviando mensagem 5
Recebi mensagem 5
Produtor enviando mensagem 6
Recebi mensagem 6
Produtor enviando mensagem 7
Recebi mensagem 7
Produtor enviando mensagem 8
Recebi mensagem 8
Produtor enviando mensagem 9
Recebi mensagem 9
Produtor enviando mensagem 10
Recebi mensagem 10
Produtor enviando mensagem 11
Produtor Terminou
Recebi mensagem 11
```