UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

E-DART - Um Ambiente de Especificação E-LOTOS

por

LISANDRO ZAMBENEDETTI GRANVILLE

Dissertação submetida à avaliação, como requisito parcial
para obtenção do grau de
Mestre em Ciência da Computação

Profª Drª Maria Janilce Bosquiroli Almeida
Orientadora

Porto Alegre, junho de 1998

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Granville, Lisandro Zambenedetti

E-DART – Um Ambiente de Especificação E-LOTOS / por Lisandro Zambenedetti Granville. – Porto Alegre: CPGCC da UFRGS, 1998.

129F.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1998. Orientadora: Almeida, Maria Janilce Bosquiroli.

1. E-LOTOS. 2. E-DART. 3. Especificação. 4. Técnicas de descrição formal. I. Almeida, Maria Janilce Bosquiroli. II. Título.

Agradecimentos

Inicialmente gostaria de agradecer a Profª Janilce pelo apoio e incentivo dispensados nestes dois anos de mestrado. Pelos puxões de orelha e pelos inestimáveis conselhos e orientação.

Da mesma forma gostaria de agradecer ao amigos Guilherme e Luciano pelo companheirismo, apoio, compreensão e principalmente pela paciência dos últimos anos. Ao pessoal do projeto DAMD, pela ajuda na definição dos caminhos do projeto e por fornecer inúmeros materiais para pesquisa; principalmente à Luciane, ao Marcelo e ao Daniel pelo inestimável empenho e dedicação dispensados.

Por fim, agradeço especialmente aos meus pais José e Maria pelo apoio e incentivo às minhas decisões, e aos meus irmãos Luciane e Leonardo pelo companheirismo e amizade.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

O aumento crescente da complexidade dos sistemas computacionais criou a necessidade do uso de técnicas de descrição formal (TDFs) na definição, implementação e manutenção dos sistemas. Contudo, apenas a existência de técnicas de descrição formal não garante o eficiente emprego das mesmas na especificação dos sistemas. É necessária a existência de ferramentas que viabilizem o uso das TDFs de forma que estas possam ser efetivamente úteis no desenvolvimento de projetos.

Este trabalho apresenta os resultados obtidos da construção de um ambiente de especificação gráfico, onde a TDF E-LOTOS (*Enhancements to LOTOS*) fornece os mecanismos que uma técnica de descrição deve possuir para viabilizar a consistência, verificação e validação de sistemas. A TDF é suportada através do emprego de uma nova versão gráfica para a sintaxe textual padrão de E-LOTOS.

O principal objetivo deste trabalho é disponibilizar aos projetistas e desenvolvedores um ambiente de especificação onde a criação, implementação, teste e manutenção de sistemas sejam feitos de forma fácil, rápida, intuitiva e consistente. Para tal, uma nova sintaxe gráfica de E-LOTOS foi criada: o E-DART (*Enhancements to DART*). Foi desenvolvida também uma ferramenta que permite o uso de E-DART, o Editor E-DART. A nova sintaxe abstrai, através de diagramas, as complexidades naturais de E-LOTOS, permitindo uma compreensão mais intuitiva dos sistemas. Com o uso de E-DART as especificações construídas são validadas por uma TDF sem que o processo de criação torne-se complexo.

No Editor E-DART, a rapidez na criação das especificações é alcançada com mecanismos de reuso de módulos, e com o emprego de recursos avançados de interação com o usuário. O reuso de módulos diminui o tempo total dispensado na criação de especificações porque as mesmas poderão utilizar partes já homologadas de outros sistemas, sem a necessidade de validação e testes, pois estes são procedimentos realizados anteriormente nas outras especificações. Os recursos avançados de interação, por outro lado, permitem que projetistas não familiarizados com E-LOTOS sejam ainda assim capazes de construir sistemas complexos. Além disso, profissionais que tenham experiência com a TDF têm acesso direto ao código textual E-LOTOS, o que lhes garante uma maior compreensão das especificações.

Palavras-chave: E-LOTOS, E-DART, Especificação, Técnicas de descrição formais

Title: “E-DART - An E-LOTOS Specification Environment”

Abstract

The increasing complexity of the computational systems has created the necessity of the use of formal description techniques (FDTs) in the systems definition, implementation and maintenance. However, only the existence of formal description techniques does not guarantee their efficient use in the systems specification. The existence of tools is necessary to make the use of the TDFFs possible in a way in which these techniques can be effectively useful in projects development.

This work presents the results of the construction of a specification graphical environment, where E-LOTOS FDT (*Enhancements to LOTOS*) supplies the mechanisms that a description technique must possess to make possible the consistency, verification and validation of systems. The FDT is supported through the use of a new graphical version for the standard textual E-LOTOS syntax.

The main goal of this work is to provide to designers and developers a specification environment where the creation, implementation, test and maintenance of systems are made in a easy form, fast, intuitive and consistent way. For such, a new graphical syntax of E-LOTOS was created: the E-DART (*Enhancements to DART*). A tool that allows the E-DART use was also developed, the E-DART Editor. The new syntax represents, through diagrams, the natural complexities of E-LOTOS, allowing a intuitive understanding of the systems. With the use of E-DART the specifications are still validated by a FDT, without turning their creation into a complex process.

In the E-DART Editor, fast creation of the specifications is achieved with mechanisms of module reuse, and with the use of advanced user interaction resources. The reuse of module diminishes the total time used in the specifications creation because it will be possible to use parts already validated of other systems. The advanced interaction resources, on the other hand, allow that designers not familiarized with E-LOTOS to be able to construct complex systems. Moreover, professionals who have experience with the FDT have direct access to textual E-LOTOS code, which guarantees an easier understanding of the specifications.

Keywords: E-LOTOS, E-DART, Specification, Formal description techniques.

1 Introdução

O aumento da complexidade dos sistemas computacionais vem crescendo dia após dia. Novas tecnologias são introduzidas no mercado constantemente, o que contribui ainda mais para esse aumento de complexidade. A definição, implementação e manutenção dos sistemas torna-se uma tarefa cada vez mais difícil e exige o uso de profissionais cada vez mais especializados. O emprego de técnicas de descrição formal procura solucionar estas dificuldades permitindo que os sistemas possam ser melhor especificados, o que diminui a complexidade dos mesmos e facilita sua implementação e manutenção. Durante os anos 80 vários esforços foram feitos no sentido de se criar uma técnica para especificar formalmente estes sistemas computacionais. Destes esforços surgiram três principais técnicas: LOTOS [BOL 87, BOL 89], SDL [SAR 87] e ESTELLE [BUD 87].

SDL, um padrão do CCITT (antigo ITU-T), rapidamente tornou-se popular no meio industrial (sobretudo nas telecomunicações) por ser de fácil compreensão e não exigir conhecimentos aprofundados sobre o assunto, dispensando assim a necessidade de profissionais especializados. Grande parte do êxito de SDL é devido à existência de uma sintaxe gráfica para a linguagem, que torna o processo de especificação bem mais simples e rápido.

Por outro lado, LOTOS e ESTELLE, que surgiram dos esforços da ISSO para a criação de uma TDF, tiveram mais êxito no meio acadêmico. LOTOS baseia-se em CCS (*Calculus of Communicating Systems*) [MIL 80] e CSP (*Communicating Sequential Processes*) [HOA 85] para a descrição comportamental dos sistemas e em ACT-ONE para a definição de tipos abstratos de dados. Apesar de uma boa base formal, tais técnicas não tiveram sucesso no meio industrial por serem de difícil compreensão. Tanto ESTELLE como LOTOS são linguagem apenas textuais o que dificultava sua compreensão. Profissionais especializados são necessários para que as linguagens possam ser utilizadas, o que dificulta muito sua disseminação. Além disso não é possível a especificação de sistemas de tempo real em LOTOS porque a TDF não possui estruturas para representar ações dependentes do tempo, fato este que limita muito o seu uso.

Nos últimos anos várias alterações e aprimoramentos foram incorporados às técnicas citadas, sempre procurando um aperfeiçoamento na forma de descrever os sistemas e tornar as linguagens mais fáceis de serem compreendidas. Várias versões de SDL, ESTELLE e LOTOS foram propostas procurando o aprimoramento das técnicas. Tais aprimoramentos introduziram novos conceitos, funcionalidades ou abstraíram construções desnecessárias.

Das três técnicas citadas, LOTOS é a mais poderosa e por essa razão a mais complexa. Com uma forte base matemática e uma sintaxe exclusivamente textual, LOTOS ainda não se apresenta como a mais popular das TDFs. Vários esforços para mudar esta situação têm sido feitos. A criação de representações gráficas busca a abstração de complexidades da sintaxe textual, enquanto que algumas extensões da linguagem textual procuram introduzir o tratamento e manipulações de ações dependentes do tempo. Sistemas multimídia, onde a noção de tempo norteia todos os aspectos do sistema, não podem ser corretamente especificados em LOTOS.

Várias extensões temporais de LOTOS foram criadas. RT-LOTOS, T-LOTOS, Tic-LOTOS são exemplos de tentativas de padronização do suporte ao tempo. Recentemente a ISO apresentou E-LOTOS (Enhancements to LOTOS) onde várias novas características foram incorporadas à linguagem, sendo uma delas o suporte ao tempo.

Esta dissertação apresenta uma proposta para tornar a linguagem E-LOTOS mais fácil de ser compreendida e utilizada. A escolha por E-LOTOS deve-se ao fato desta TDF ser a mais poderosa existente. Busca-se uma maior disseminação da mesma através do uso de uma ferramenta de especificação. O objetivo principal é disponibilizar aos usuários que especificam sistemas todo o poder de E-LOTOS de uma forma simples e intuitiva, abstraindo as principais complexidades da linguagem. Para tal, o presente trabalho introduz uma nova sintaxe gráfica: o E-DART. A sintaxe gráfica possui suporte ao tempo e é capaz de representar grande parte das construções LOTOS. Além de E-DART foi desenvolvido a ferramenta Editor E-DART que suporta a nova sintaxe. Com este ambiente os usuários poderão especificar sistemas utilizando gráficos intuitivos, além de terem ao mesmo contato com a sintaxe textual [GRA 98A, GRA 98B, GRA 98C, GRA 98D, GRA 98E, GRA 98F].

O capítulo 2 apresenta um estudo sobre a necessidade da existência de extensões temporais para LOTOS, além de descrever algumas destas principais extensões. O capítulo 3 introduz a nova TDF E-LOTOS da ISO. O capítulo 4 faz um estudo e comparação entre duas sintaxes gráficas de LOTOS: G-LOTOS e DART. Tais sintaxes foram a base para a criação de E-DART, a nova sintaxe gráfica suportada pela ferramenta criada. O capítulo 4 introduz E-DART comparando-o com as propostas gráficas apresentadas no capítulo 3. Por fim, o capítulo 5 discorre sobre o Editor E-DART, seu projeto, implementação e principais características.

2 Extensões Temporais de LOTOS

LOTOS é uma técnica de descrição formal que possui uma sintaxe textual rígida e matemática. Isto garante especificações geradas sobre regras precisas as quais pode-se aplicar validações. Tais validações só são possíveis devido a notação matemática existente. Assim, os sistemas especificados são bem definidos mas muitas vezes algumas características não podem ser modeladas. Sistemas de tempo real, por exemplo, não podem ser definidos em LOTOS. Isto porque não existe a noção de tempo quantitativo nas especificações.

Os sistemas em LOTOS são definidos através de um conjunto de processos que trocam informações entre si. As ações mais críticas em LOTOS são as chamadas sincronizações. Tais ações ocorrem quando um determinado processo LOTOS, através de canais de sincronização, comunica-se com outro processo. Esta comunicação entre dois processos LOTOS é chamada de sincronização. Se determinada sincronização não é possível, então diz-se que os processos envolvidos terminaram sem sucesso. Uma sincronização não tem duração e é dita instantânea. Entretanto, tais sincronizações instantâneas não acontecem na prática, quando as especificações LOTOS tornam-se sistemas implementados, pois as comunicações entre processos muitas vezes duram um tempo considerável até seu término.

A expressão "extensão temporal" refere-se à um incremento na linguagem LOTOS que possibilite a manipulação e o tratamento do tempo nas especificações, o que permite a criação de sistemas de tempo real, por exemplo. Várias extensões temporais foram propostas: Temporal LOTOS [REG 93], TIC-LOTOS [AZC 89], T-LOTOS [BOL 93], RT-LOTOS [QUE 87] e E-LOTOS [ISO 97] são alguns exemplos. O objetivo principal de todas elas é fornecer um mecanismo capaz de medir o tempo decorrido nos eventos LOTOS. Desta forma, as sincronizações passam a ser não instantâneas, e o tempo decorrente em uma sincronização pode ser mensurado. De uma forma geral as principais construções incorporadas à LOTOS pelas extensões temporais são:

- Operador capaz de indicar o tempo decorrido de uma sincronização. Com tal operador é possível determinar quanto tempo uma sincronização pode levar. É possível implementar-se políticas de *timeout* e de tempo mínimo de sincronização;
- Operador de atraso. O operador de atraso suspende o processo por um determinado período de tempo, permitindo assim um retardo na sua execução.

Agora mostraremos de forma mais específica a necessidade de existência das extensões temporais. Como vimos anteriormente, o tempo não pode ser medido nos sistemas LOTOS. Isso, entretanto, não significa que tais sistemas não possam prever a ocorrência de eventos que denotem, por exemplo, o *timeout* de uma operação. Isso pode ser feito através do operador de ação interna ***i***. A questão, neste caso, é saber se o operador de ação interna é suficiente para expressar o tempo. [REG 93] apresenta a seguinte explanação provando que apenas o operador de ação interna mostra-se insuficiente.

2.1 Operador de Ação Interna

Para exemplificarmos o uso do operador de ação interna em eventos que envolvam a passagem do tempo usaremos um exemplo. Assim, o processo *TESTER* é definido como segue:

```
TESTER := send;(receive;success;exit [] i;TESTER)
```

A função do processo é testar a confiabilidade de um ambiente de comunicação. Primeiro, uma mensagem é enviada através deste meio (denotado pela ação *send*). Então, se o ambiente testado retornar uma mensagem (denotado pela ação *receive*) o processo aceita a mensagem de resposta e sinaliza o sucesso do teste (denotado pela ação *success*) e então finaliza. Caso a mensagem de retorno não for recebida, ocorre um *timeout* (evento representado então pelo operador da ação interna ***i*)** e o processo de teste é reiniciado. Este reinicio ocorre até que o ambiente testado responda de forma correta ao processo.

Vamos considerar agora um processo que modela um ambiente de comunicação não confiável, denotado pelo processo a seguir.

```
MEDIUM := send;(receive;MEDIUM [] i;MEDIUM)
```

O processo *MEDIUM* receberá primeiramente uma mensagem (*send*) e então procederá de uma das seguintes formas: retornará uma resposta (*receive*) ou internamente perderá a mensagem. Para modelar os testes do ambiente de comunicação temos a seguinte definição.

```
hide send,receive in TESTER |[send,receive]| MEDIUM
```

Para a construção acima temos o seguinte gráfico de ações executadas:

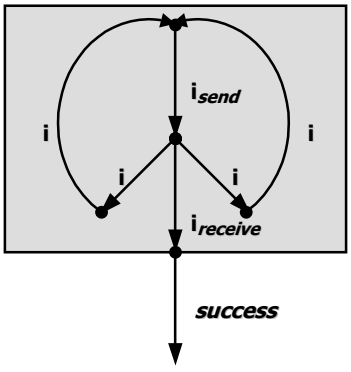


FIGURA 2.1 - Ações de um sistema divergente

Tal processo é similar à construção: `i;success;exit`. Aqui podemos notar uma primeira anomalia em relação ao tratamento de *timeout*. O sistema, como visto no gráfico, é divergente. Mas este fato é ignorado quando observamos as trocas de mensagens como ações internas. Isso é facilmente notado se analisamos a construção similar apresentada para o sistema. Nesta caso chega-se a conclusão que o tratamento do tempo através de ações internas é uma abstração que esconde detalhes muito importantes, como no caso a divergência do sistema.

Consideremos agora um ambiente confiável em nosso teste. A definição deste ambiente confiável é a seguinte:

```
MEDIUM := send;receive;MEDIUM
```

Neste caso teremos o seguinte gráfico de ações executadas:

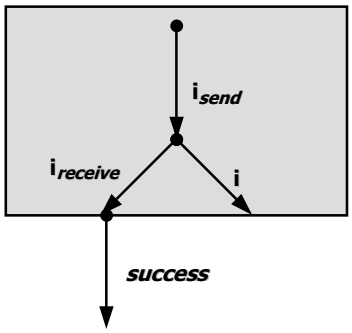


FIGURA 2.2 - Ações de um sistema com *deadlock*

Este processo por sua vez é similar a: `i;success;exit [] i;stop`. Neste caso, a introdução de um ambiente de teste confiável acarretou o surgimento de um *deadlock* (o sistema pode resultar em um *stop*) errôneo. Isto porque a operação ***i*** não possui prioridade sobre a ação *receive*. Assim, a ação interna modela uma *timeout* indefinido, o que não é aceitável.

Com estes dois exemplos fica demonstrado que o uso de ações internas para o tratamento do tempo não é adequado, ficando necessária a introdução de construções específicas e explícitas para a solução do problema. A criação de tais construções é o objetivo final das extensões temporais aqui apresentadas.

2.2 Temporal LOTOS

Temporal LOTOS [REG 93] é baseado em TPL [HEN 91, REG 91], extensão temporal de CCS. O tempo é introduzido de forma mínima, isto é, apenas a dimensão do tempo e uma ação denotada pelo símbolo σ, que representa um *tick* de relógio, foram introduzidos. Assim, pode-se prefixar um processo com essa ação para se representar um atraso unitário imposto a um processo. Nenhuma estrutura sintática que permita especificar o tempo de forma explícita e quantitativamente existe em Temporal LOTOS.

Para exemplificar o uso da construção σ vamos definir o processo *TESTER,* da seção anterior como sendo:

```
TESTER := send;(receive;success [] σ;TESTER)
```

Aqui, σ foi usado para representar a passagem do tempo. Esta ação não será disparada se existir qualquer ação interna possível de ser avaliada. Neste caso teremos o seguinte gráfico de ações executadas.

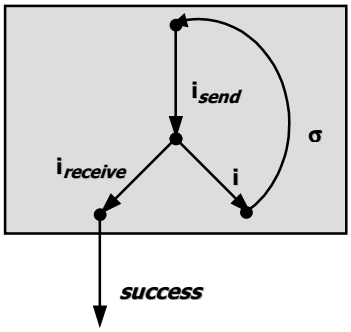


FIGURA 2.3 - Ações com o operador **σ**

Aqui, a divergência antes existente foi substituída pela representação explícita da passagem do tempo. Isto é equivalente ao processo: `P := i;success;exit [] i;σ;P`.

Se quisermos um teste com um ambiente confiável teremos então o seguinte gráfico de ações.

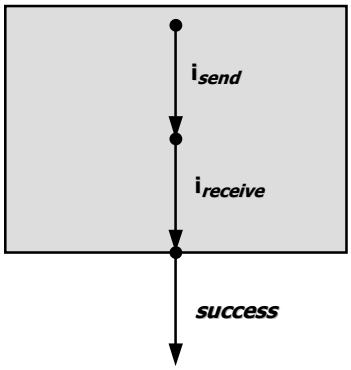


FIGURA 2.4 - Ações para um ambiente confiável

Neste caso, isto é equivalente ao processo `i;success;exit`.

A linguagem é definida como uma extensão estrita de LOTOS, e apenas um novo operador é definido: o operador "then". Tal operador é escrito na forma ⌊-⌋ (-). Por exemplo, o comportamento para a expressão

```
⌊send_fax;BACK_TO_WORK⌋ (post_letter;GO_HOME)
```

é tentar inicialmente enviar um fax (*send_fax*) e retorna ao trabalho (*BACK_TO_WORK*). Entretanto, se isto não for possível no momento, então a ação será tentar enviar uma carta (*post_letter*) e ir para casa (*GO_HOME*).

Assim, para ⌊P⌋(Q), inicialmente as ações de *P* são oferecidas. Se uma delas for realizada teremos então uma transição $P \rightarrow P'$. Neste caso o comportamento ⌊P⌋(Q) evoluirá para *P'*, isto é ⌊P⌋(Q) → P'. Isto desabilitará a execução de *Q*. Porém, se nenhuma das ações de *P* ocorrer em uma unidade de tempo, então o processo *P* é abandonado e o processo *Q* é executado.

Por fim, pelo fato de o único operador incluído por Temporal LOTOS ser o operador "then", temos a inexistência de operadores de bloqueio de tempo, isto é, operadores capazes de retardar o progresso do tempo.

2.3 TIC-LOTOS

Timed-Calculus for LOTOS [AZC 89], ou simplesmente TIC-LOTOS, é uma evolução de trabalhos de diversos autores da Universidade de Madri [QUE 87]. Nesta proposta temos a adição de um modelo de tempo visando a representação de processos temporizados. Isto é alcançado pela introdução da noção de tempo quantitativo através de restrições sobre a ocorrência de eventos.

O tempo em TIC-LOTOS é representado por valores inteiros positivos e globais. Isto é, a passagem do tempo é "vista" por todos os processos de um sistema da mesma forma. As expressões de comportamento em TIC-LOTOS geram sistemas de transição rotulados. Tais rótulos são os eventos do sistema temporizados. Cada rótulo é um contador de tempo relativo às transições anteriores. Isso eqüivale a existência de um relógio global implícito no sistema de transição.

A seguir são enumeradas as principais características acrescentadas a LOTOS por TIC-LOTOS, que são:

1. Uma etiqueta de tempo é atribuída a cada ação para indicar o tempo exato em que ocorreu;
2. As ações são autônomas, isto é, ocorrem no tempo *timestamp* que lhe é associado e;
3. Um construtor de escolha temporal ("time choice") é introduzido como extensão do operador de prefixação para representar a possibilidade de realização de uma ação dentro de um intervalo de tempo.

Assim, cada evento ocorre em um dado instante de tempo definido. A figura abaixo exemplifica a passagem do tempo em TIC-LOTOS para três expressões de comportamento.

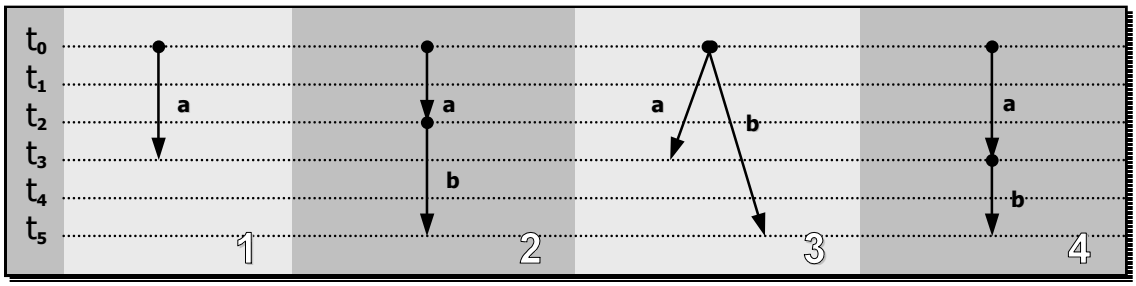


FIGURA 2.5 - Exemplos em TIC-LOTOS

Em (1) temos o evento *a* ocorrendo no instante de tempo 3. Logo, para (1) temos como expressão TIC-LOTOS a construção seguinte:

```
a3
```

Em (2) o evento *a* ocorrerá no tempo 2 e o evento *b* no tempo 5, que corresponde a três unidades de tempo após a execução de *a*. Neste caso, visto que o tempo especificado para cada evento corresponde ao instante em que o mesmo ocorrerá **após** a execução do evento anterior, temos a construção:

```
a2;b3
```

Em (3), admitindo que após *a* e *b* temos o comportamento LOTOS *stop*, a construção a seguir representa tal comportamento.

```
a3;stop ||| b5;stop
```

Por outro lado, este comportamento pode ser expresso também por:

```
a3;b2;stop
```

Esta construção também define o gráfico (4). Logo, temos que os comportamento em (3) e (4) são equivalentes de acordo com TIC-LOTOS.

Por fim, cabe salientar que, assim como ocorria em Temporal LOTOS, TIC-LOTOS não possui construções para indicar o atraso explícito nos processos, isto é, o operador de atraso, citado anteriormente, também não está presente em TIC-LOTOS.

2.4 T-LOTOS

T-LOTOS (*Timed LOTOS*) [] é o modelo no qual as ações relacionadas ao tempo têm um caráter preemptivo sobre as demais. Adota-se uma semântica de árvores com ortogonalidade de tempo/ação e as transições devidas às ações não decorrentes da passagem do tempo em T-LOTOS tem a forma a seguir.

$$B \xrightarrow{a^t} B'$$

Aqui, o comportamento *B* realiza a ação *a* de idade *t* e transforma-se no comportamento *B'*. O tempo de oferecimento de uma ação é incrementado por transições de passagem de tempo. Assim, em a^t , o *t* registra exatamente o tempo em que *a* foi oferecido.

T-LOTOS acrescenta ainda dois operadores temporais primitivos à linguagem: "time" e "time-once". Para o operador "time" temos a seguinte construção.

```
time a (t1,t2) in B
```

Aqui, é permitida a operação *a*, no processo *B*, ocorrer em um intervalo de tempo limitado por *t1* e *t2*. Assim, *a* deve ocorrer no intervalor (*t1,t2*) a menos que seja desabilitada pela ocorrência de outra ação. Para o operador "time-once" temos a seguinte construção.

```
time-once a (t1,t2) in B
```

Este operador possui uma semântica quase idêntica ao operador "time". A única diferença é que ele perde seu efeito após uma primeira aplicação. Com estes dois operadores primitivos pode-se definir um conjunto grande de operadores derivados que permitem expressar funcionalidades bastante interessantes. Desta forma, temos os seguintes operadores derivados de "time" e "time-once" para T-LOTOS.

TABELA 2.1 - Operadores T-LOTOS

Operador	Definição	Significado
`a(t1,t2); B`	`time-once a(t1,t2) in B`	A operação *a* ocorrerá em *B* no intervalo (*t1,t2*) se não for desabilitada por outra operação.
`a(t); B`	`a(t,t); B`	A operação *a* ocorrerá em *B* após um retardo de *t* unidades de tempo.
`urge a in B`	`time a(t1,t2) in B`	A operação *a* deverá ocorrer em *B* assim que ele esteja disponível, não devendo ser retardada sua execução.

Em T-LOTOS o conceito de urgência, que determina se uma ação deve ser avaliada assim que disponível, é bem definido. As ações quando especificadas são, normalmente, não urgentes. Porém, elas podem tornar-se urgentes se forem submetidas a um operador que imponha este tipo de comportamento. Todas essas funcionalidades podem ser alcançadas com a utilização dos operadores apresentados anteriormente.

2.5 ET-LOTOS

ET-LOTOS (*Enhanced Timed LOTOS*) [ ] introduz dois operadores básicos para o tratamento do tempo na linguagem: o operador de prefixação temporal e operador de atraso. O operador de prefixação temporal possui a forma geral a seguir.

```
a@t{d};E
```

Neste operador as construções *@t* e *{d}* são opcionais. A semântica para o operador é definida pelas regras seguintes.

O operador *{d}* representa o conceito de redutor de vida que quando operado com a ação *a* atribui à prefixação a seguinte semântica: a ação *a* é oferecida durante um período de tempo *d* durante o qual ela pode se realizar. Após este período, se *a* não se realizou, então o processo termina sem sucesso transformando-se no comportamento *stop.* Consideremos assim o seguinte exemplo.

```
send {30}; OK
```

A ação *send* é oferecida ao ambiente e deve ser realizada no máximo em 30 unidades de tempo. Caso isso acontece teremos a evolução para a construção *OK*. Caso contrário, *OK* não será avaliado e o processo evoluirá para *stop*, indicando o insucesso da ação *send*.

A construção *@t* representa o registro do tempo de espera até a realização da ação que está associada a ele. Para exemplificar tomemos a expressão a seguir

```
a@t;E
```

Aqui, a construção indica que o tempo relativo no qual a ação *a* começou a ser oferecida é registrado na variável *t*, que será utilizada por *E*.

Em ET-LOTOS o operador de atraso permite a imposição de um retardo na evolução de um processo e é denotado por

```
Δ[d1,d2]
```

O atraso é definido por um tempo não determinístico dentro do intervalo entre *d1* e *d2*. Assim, em ET-LOTOS é possível proceder com a imposição de atrasos nos processos das especificações, permitindo modelar-se mais fielmente diversas condições de evolução do tempo nos sistemas especificados.

3 A Nova Extensão E-LOTOS

E-LOTOS (*Enhancements to LOTOS*) [ISO 97] é resultado dos esforços da ISO na revisão da norma que define LOTOS. Atualmente, E-LOTOS encontra-se na forma de *Draft Proposal* mas grandes contribuições na melhoria da linguagem já foram definidas.

As melhorias incorporadas a linguagem estão, em sua maioria, relacionadas com a introdução da noção de tempo quantitativo e com a redefinição da forma de representação dos dados, de forma a tornar a definição dos mesmos mais amigável e concisa.

Como mencionado anteriormente, a introdução de estruturas que suportem a noção de tempo quantitativo é necessária para possibilitar a descrição de sistemas em tempo real. Já as alterações na definição dos tipos de dados permitem que os mesmos possam ser descritos em dois níveis distintos. Num primeiro nível tem-se ainda a definição através de ACT-ONE [EHR 85], como era feito em LOTOS. O segundo nível, por sua vez, possui uma semântica operacional, o que denota uma abordagem funcional, suportando tipos de dados pré-definidos, registros definidos pelo usuário, subtipos de registros, união e recursão.

Além do mencionado anteriormente, E-LOTOS apresenta ainda as seguintes melhorias:

- Tipificação explícita de portas na declaração dos processos;
- Introdução de estruturas para modularização das especificações: módulos, módulos genéricos e interfaces;
- Definição de novos operadores, como “case” e “loop”, simplificando a linguagem;
- Introdução de estruturas para a definição e tratamento de exceções.

E-LOTOS é constituído de dois níveis, onde o nível superior trata-se da linguagem modular (*Modular Language*), e é responsável pela modularização das especificações. Já o nível inferior constitui-se na linguagem base (*Base Language*) que concentra toda a base formal de E-LOTOS.

A seguir são apresentadas as estruturas da linguagem base e da linguagem modular que foram usadas neste trabalho.

3.1 A Linguagem Base

As especificações criadas com a linguagem base são constituídas de uma seqüência de declarações. A frente veremos que estas declarações podem ser organizadas em módulos pela linguagem modular.

Uma declaração na linguagem base pode ser de três tipos: declaração de tipos, declaração de funções e declaração de processos.

Declaração de Tipos

Uma declaração de tipo é tanto a declaração de um *tipo sinônimo* ou a declaração de um *tipo de dado*. Uma sinônimo declara um novo identificador para um tipo já existente.

```
type <identificador de tipo> is
     <expressão de tipo>
endtype
```

Por exemplo, podemos declarar o tipo "point" como sendo um sinônimo para um *registro* de tipos "float".

```
type point is
   (x ⇒ float, y ⇒ float)
endtype
```

Já a declaração de um tipo de dado, define um novo tipo para a especificação sendo construída.

```
type <identificador de tipo> is
     <identificador de construtor> [(<registro de expressão de tipo>)]
  (| <identificador de construtor> [(<registro de expressão de tipo>)])*
endtype
```

Assim, podemos declarar, por exemplo, uma lista de inteiros recursiva, como segue.

```
type intlist is
   nil
 | cons (int, intlist)
endtype
```

A declaração de um tipo de dado lista todos os *construtores* para o tipo. Visto que podemos ter mais de um construtor, podemos usar *uniões*, como no exemplo a

seguir, onde o tipo “pdu” representa uma mensagem “send” ou “ack”, com seus respectivos parâmetros.

```
type pdu is
   send (packet, bit)
 | ack (bit)
endtype
```

Tipos de dados que são definidos como registros podem ser extensíveis através do tipo “etc”. Assim a construção

```
(name ⇒ string, etc)
```

é um registro de pelo menos um campo “name”, mas que pode ser estendido para possuir outros campos.

Além de “etc”, E-LOTOS possui ainda mais dois tipos especiais:

- O tipo vazio “none” sem valores, usado para definir a funcionalidade de processos que nunca terminam.
- O tipo universal “any”, que é o supertipo para todos os outro tipos, usado para definir o tipo “default” das portas de comunicação dos processos. Portas do tipo “any” pode trocar dados de qualquer tipo, o que garante a compatibilidade com LOTOS.

Outra característica dos tipos é que a igualdade entre eles é dada *estruturalmente* e não *nominalmente*. Os tipos “colpixel” e “colpixel2”, a seguir, podem ser usados igualmente. Assim, uma função que recebe como parâmetro um “colpixel” aceitará um “colpixel2” da mesma forma.

```
type colpixel is
   (pt ⇒ point, col ⇒ colour)
endtype

type colpixel2 is
   (pt ⇒ (x ⇒ float, y ⇒ float), col ⇒ colour)
endtype
```

A linguagem base não fornece um mecanismo para definição de tipos parametrizados. Entretanto, isto é possível através da linguagem modular.

Declaração de Funções

Em E-LOTOS uma função é tratada como um processo que executa somente uma ação de terminação. Assim, quando a ação de terminação ocorrer os dados resultantes são passados como resultado da função ao processo no qual a função está envolvida.

A declaração de uma função define uma nova função, que pode ser usada em expressões de dados.

```
function <identificador de função> [(<variáveis locais>)] [: <expressão de tipo>]
   [ raises [<identificador de exceção> [: <expressão de tipo>]
          (, <identificador de exceção> [: <expressão de tipo>])*] ]
   is E
endfun
```

Uma função pode ter mais de um parâmetro de entrada, e pode retornar um registro como resultado, por exemplo, no caso da função “partition”:

```
function partition (x:int, xs: intlist) : (intlist, intlist) is
   local
      var less: intlist, gtr: intlist
   init
      ?less := todos os x's menores que x;
      ?gtr := todos os x's maiores que x
   in
      (less, gtr)
   endloc
endfun
```

Esta função pode ser chamada como segue, na função “quicksort”:

```
function quicksort (xs: intlist) : intlist is
   case xs is
      nil →
         nil
    | cons (?y, ?ys) →
         local
            var l: intlist, g: intlist
         init
```

```
            (?l, ?g) := partition (y, ys)
         in
            append (quicksort (l), cons(y, quicksort (g)) )
         endloc
      endcase
   endfun
```

Este estilo de função é muito usado. Entretanto, E-LOTOS fornece uma segunda opção para declaração de funções usando parâmetros de entrada e saída.

```
function<identificador de função>([in<variáveis locais>)][out<variáveis locais>])
   [ raises [<identificador de exceção> [: <expressão de tipo>]
          (, <identificador de exceção> [: <expressão de tipo>])*] ]
   is E
endfun
```

A função “partition” poderia ser assim escrita da seguinte forma:

```
function partition (in x: int, xs: intlist,
                    out less: intlist, gtr: intlist) is
   ?less := todos os x's menores que x;
   ?gtr := todos os x's maiores que x
endfun
```

e usada na função “quicksort” como:

```
partition (y, ys, ?l, ?g)
```

no lugar de:

```
(?l, ?g) := partition (y, ys)
```

Declaração de Processos

A declaração de processos é muito similar a declaração de funções: os processos possuem listas de parâmetros, parâmetros de entrada e saída, tipo de saída (indicado pela palavra reservada “exit”). Entretanto, existem duas importantes diferenças entre processos e funções: processos podem ter comportamento de tempo real, e podem comunicar-se através de portas.

Um processo pode ser declarado também de duas formas distintas:

```
process<identificador de processo> [[[<identificador de porta>[:<expressão de tipo>]
        (,<identificador de porta>[:<expressão de tipo>])*]]]
        [(<variáveis locais>)]: exit (<expressão de tipo>)
    [raises [<identificador de exceção> [:<expressão de tipo>]
            (,<identificador de exceção> [:<expressão de tipo>])*]]
    is B
endproc
```

ou:

```
process<identificador de processo> [[[<identificador de porta>[:<expressão de tipo>]
        (,<identificador de porta>[:<expressão de tipo>])*]]]
        ([in <variáveis locais>] [out <variáveis locais>])
    [raises [<identificador de exceção> [:<expressão de tipo>]
            (,<identificador de exceção> [:<expressão de tipo>])*]]
    is B
endproc
```

Por exemplo, a seguinte declaração define um processo simples “Counter”:

```
process Counter [up: (), down ()] is
    up; (down ||| Counter [up, down])
endproc
```

Neste caso, o processo comunica-se através das portas “up” e “down”. Por “default”, o tipo de uma porta é “any”, o que permite a comunicação de dados de qualquer natureza. Isso garante a compatibilidade com especificações LOTOS, onde as portas de comunicação não eram tipadas.

A definição dos processos constitui-se no foco central de uma especificação, pois são os processos que definem o comportamento dinâmico das especificações. A seguir são apresentadas as construções utilizadas na definição do comportamento dos processos.

3.2 Expressões de Comportamento

As expressões de comportamento definem a ordem temporal em que os eventos de uma especificação E-LOTOS ocorrerão. Assim, um sistema é definido através de um conjunto de expressões de comportamento e um conjunto de operadores, associados a estas expressões. A seguir são apresentadas as expressões de comportamento mais relevantes que foram utilizadas neste trabalho.

Inação e Bloqueio do Tempo

A expressão **stop** representa um comportamento completamente inativo, onde nenhuma ação pode ser oferecida ao ambiente exceto a passagem do tempo. Já a expressão **block** além de representar um comportamento inativo não permite também a passagem do tempo.

Terminação

Um processo em E-LOTOS termina com sucesso sempre que a expressão **exit** ocorrer. Diz-se então que o processo terminou com sucesso, ao passo que a expressão **stop** representa um *dead-lock* e diz-se que o processo terminou sem sucesso.

A terminação pode ser determinística, onde **exit** informa explicitamente o resultado do processo.

```
exit [(<registro de expressão de valor>)]
```

Já a terminação não determinística permite o retorno de qualquer expressão de tipo de dados.

```
exit (any <expressão de tipo>)
```

Após a ocorrência de um **exit** a expressão de comportamento torna-se **block**, impedindo a continuação do processo envolvido.

Composição Seqüencial

A composição seqüencial define a ordem de ocorrência de dois eventos. Denotado pelo símbolo de ponto-e-vírgula (;) define o término de um evento e o início do próximo. Assim, o que inicia um comportamento será sempre a ação de terminação do comportamento anterior. Assim:

```
?t := 3; exit
```

é uma seqüência, onde ocorrerá primeiramente uma atribuição e logo a seguir o comportamento **exit**.

Atribuições

O comportamento `P := E` é uma expressão imperativa, onde *P* são os parâmetros da atribuição, enquanto *E* é uma expressão de dados que será associada aos parâmetros de *P*.

```
(?l, ?r) := reflect (pt)
```

Neste caso o resultado da função "reflect" será atribuído às variáveis "l" e "r". Variáveis em E-LOTOS são ditas de escrita única (*write-once*), onde isto se deve ao fato de que as regras semânticas estáticas da linguagem garantem um uso seguro de variáveis somente neste caso. Este tema encontra-se ainda em estudo pela ISO, não estando a questão fechada sobre variáveis de escrita única.

A expressão

```
?x:=0; ?y:=0; exit
```

tem o mesmo comportamento que

```
exit (x⇒0, y⇒1)
```

É possível ainda fazer atribuições não determinísticas, onde o valor atribuído às variáveis é aleatório. Neste caso a atribuição é combinada com "any". Assim temos:

<*parâmetros de atribuição*> := any <*expressão de tipo*> [<*expressão de dados*>]

Exemplo: `?x := any int` onde será atribuído a x um valor inteiro aleatório.

Ação Interna

A ação interna é denotada pela letra *i*, e representa um comportamento que não altera o contexto do processo, pois não há uma associação de valores com sua terminação. A ação interna é utiliza para especificar a abstração de uma ação que pode ser, por exemplo, definida na implementação da especificação por uma linguagem de programação.

Ação Observável

Uma ação observável é definida da seguinte forma:

```
<identificador de porta> [<parâmetros>] [@ <parâmetros>]
[[<expressão de dados>]] [start (<expressão de valor>)]
```

ou simplesmente: `G[P1][@P2][[E]][start(N)]`

E é uma expressão booleana que considera em seu contexto *P1* e *P2*, onde *P1* são os chamados parâmetros de encaixe de dados, e *P2* os parâmetros relativos à passagem de tempo durante a ação. Assim, diz-se que a ocorrência de uma ação associada à porta *G* implica no encaixe dos valores de *P1* e *P2*, além da avaliação da expressão *E* como sendo verdadeira, após *N* unidades de tempo a partir da habilitação da ação. A não satisfação de alguma destas condições implica na não ocorrência da ação associada a *G*, resultando num comportamento **stop**.

Quando os elementos opcionais não forem usados temos os seguintes valores: P1=(), P2=@any, E=[true] e start(0).

Com estas construções é possível implementar políticas de *time-out* nas sincronizações, além de definir o escopo das ações nas portas de sincronização. Assim:

```
G(nome⇒?n, idade⇒!21, sexo⇒!"masc") @ ?t [t≤3]
```

a porta *G* será sincronizada com qualquer pessoa do sexo masculino, e que possua 21 anos, onde a variável n será associada a um valor apropriado, sendo que esta ação poderá ocorrer em até 3 unidades de tempo, onde o tempo desde a habilitação até o sincronismo é armazenado na variável *t*.

Atraso (delay)

O comportamento de atraso permite a introdução de um atraso de um número definido de unidades de tempo, resultante da avaliação de uma expressão de dados, ou de um número não determinístico destas unidades, através da associação com "any".

O operador de atraso wait e os parâmetros existentes nas ações observáveis (sincronizações nas portas dos processos) foram introduzidos em E-LOTOS como suporte à noção de tempo quantitativo. Os seguintes exemplos mostram como o operador "wait" pode ser utilizado nas especificações.

```
wait(3); i; exit
```

Neste caso haverá um atraso de 3 unidades de tempo na avaliação da ação interna ***i***. A passagem de tempo irá decrementar 3 até que o comportamento torne-se

`wait(0)` habilitando assim a ação ***i***. Para um atraso não determinístico de ***i*** temos a seguinte expressão:

```
wait(any time); i; exit
```

Desabilitação (preempção)

A expressão de desabilitação permite que um comportamento possa ser desabilitado por outro comportamento, no momento que uma ação for tomada por este último. Assim:

```
B1 [> B2
```

B2 pode desabilitar o comportamento *B1* desde que ocorra uma ação em *B2*. Entretanto, a ocorrência da ação de terminação de *B1* encerra o comportamento de desabilitação, impedindo assim que *B2* possa ainda ser executado. Tanto *B1* quanto *B2* devem resultar tipos idênticos para que a expressão desabilitação seja um comportamento válido.

Escolha

Para o comportamento a seguir:

```
B1 [] B2
```

se uma ação ocorrer em *B1* ou *B2* então o comportamento resultante da escolha será igual ao comportamento da expressão na qual ocorreu a ação, resolvendo assim a escolha. Porém, a passagem do tempo, que deve ser observada simultaneamente nos comportamentos envolvidos, não resolverá uma expressão de escolha. Por exemplo:

```
a; P1[…] [] wait(3); b; P2[…]
```

A ocorrência de *a* resolve a escolha, e *P1* será a próxima ação tomada no comportamento. Entretanto, a passagens de 3 unidades de tempo habilita a execução de *b*, mas não resolve a escolha. Por exemplo, se na segunda unidade de tempo *a* ocorrer, então *b* não poderá mais ocorrer apesar de *wait(3)* ter sido iniciado, visto que este último por si só não resolve a escolha.

Composição Paralela

Existem três forma de expressar a composição paralela, ou concorrência, entre expressões de comportamento. Comportamentos totalmente paralelos, isto é, onde não há nenhuma sincronização entre os comportamentos, possuem a seguinte sintaxe:

```
<expressão de comportamento> ||| <expressão de comportamento>
```

Aqui os dois comportamentos são avaliados independentemente e em paralelo, sem trocarem informações ou executarem sincronizações. Este tipo de comportamento também é conhecido como *entrelaçamento.* A *sincronização parcial* é dada por:

```
<expressão de comportamento>
|[[<identificador de porta> (,<identificador de porta>)*]]|
<expressão de comportamento>
```

onde as expressões de comportamento sincronizam-se através das portas indicadas. Por fim tem-se a *sincronização completa*, onde as expressões de comportamento sincronizam-se em todas as portas envolvidas no comportamento, e é dada por:

```
<expressão de comportamento> || <expressão de comportamento>
```

Ocultação

O operador "hide" é responsável por implementar ocultação em E-LOTOS, possuindo a seguinte sintaxe:

```
hide <identificador de porta> [(<expressões de tipo>)]
    (,<identificador de porta> [(<expressões de tipo>)])*
    in <expressão de comportamento>
endhide
```

As portas listadas são ocultadas do comportamento, transformando-se em ações internas não observáveis.

Declaração de Variáveis Locais

As variáveis locais são definidas pelo operador “local”, que restringe o escopo das variáveis dentro de sua estrutura.

```
local
   var less: intlist, gtr: intlist
init
   less := smaller (xs, x);
   gtr := greater (xs, x);
in
   B
endloc
```

No exemplo, *less* e *gtr* são válidas apenas dentro do escopo da expressão de comportamento *B*. A seção “init” é utilizada para inicialização opcional das variáveis.

3.3 A Linguagem Modular

A linguagem modular foi introduzida por E-LOTOS para permitir uma melhor organização das especificações, bem como implementar conceitos de orientação a objeto nas especificações. De forma geral, um sistema especificado em E-LOTOS é uma seqüência de declarações do tipo:

- Declaração de Módulos;
- Declaração de Interfaces; e
- Declaração de Módulos Genéricos

Com estas construções, é possível a importação, exportação, ocultação e generalização dos módulos de uma especificação. Do ponto de vista da abstração e reutilização de código, estas estruturas são aquelas que permitem tais construções.

Este trabalho utilizou, numa primeira instância, apenas o conceito de módulos, deixando a definição das interfaces e dos módulos genéricos como trabalhos futuros. Apesar deste escolha, os conceitos de abstração e reuso de código não foram prejudicados, pois os mesmos foram alcançados por outros meios, discutidos mais a frente.

A seguir apresentamos a definição de módulos usada no trabalho.

Declaração de Módulos

Um módulo em E-LOTOS é uma seqüência de declarações de tipos, construtores, valores constantes, funções e processos, utilizando-se de todos os recursos da linguagem base. Assim, um módulo pode ser declarado da seguinte forma:

```
module <identificador de módulo> is <declarações> endmod
```

onde as declarações são feitas através de construções da linguagem base. Com isto, temos que uma especificação em E-LOTOS pode ser vista como o seguinte esquema.

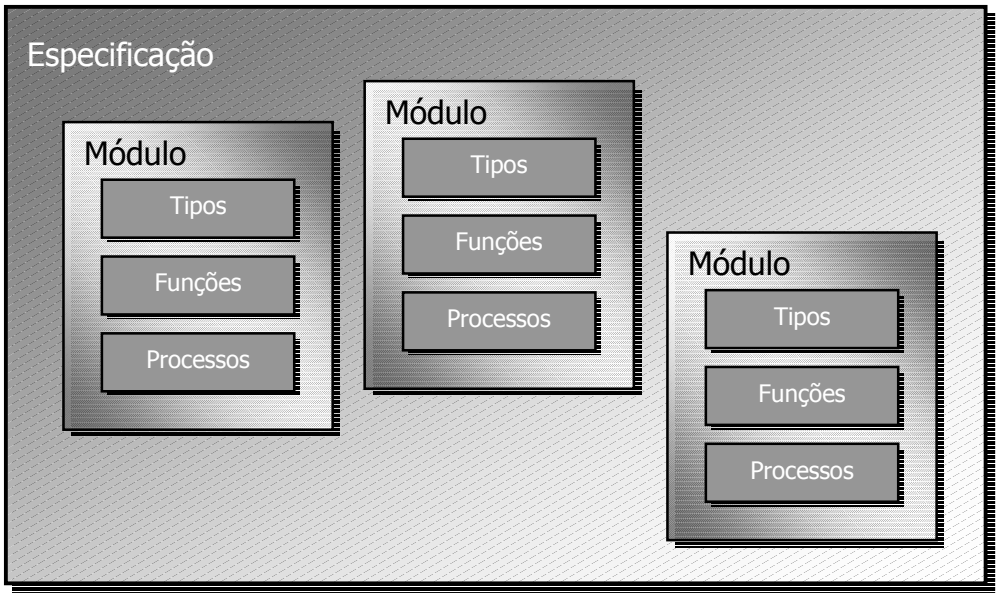


FIGURA 3.1 - Esquema de uma especificação E-LOTOS

4 Sintaxes Gráficas para LOTOS

A necessidade de uma sintaxe gráfica para LOTOS surgiu do fato de a sintaxe textual possuir uma notação matemática complexa, o que dificultava em muito o uso de LOTOS. Era necessário o uso de profissionais especializados na especificação de sistemas, o que muitas vezes era inviável economicamente. Desta forma, LOTOS não era acessível a muitos profissionais envolvidos na especificação de sistemas.

As sintaxes gráficas procuram facilitar a interatividade com a linguagem, abstraindo complexidades da sintaxe textual e introduzindo estruturas mais intuitivas ao usuário final. A eficiência do emprego de sintaxes gráficas pode ser observada, por exemplo, em SDL. A TDF possui tanto uma representação textual quanto gráfica, mas a grande maioria dos sistemas é especificado através de diagramas, por estes serem mais fáceis de serem compreendidos.

Os motivos para o uso de uma sintaxe gráfica para LOTOS, no lugar da sintaxe textual, podem ser resumidos nos seguintes itens:

- Livrar o usuário de preocupações com a correção sintática das especificações;
- Apresentar uma forma de especificação mais próxima dos conhecimentos já adquiridos pelo usuário.

LOTOS possui uma sintaxe com regras bem definidas e obriga, desta forma, a construção de especificações sintaticamente corretas, livres de inconsistências. Entretanto, através da sintaxe textual, muito tempo é perdido pelo usuário para garantir a correção sintática das especificações. Um símbolo de ponto-e-vírgula ausente, por exemplo, pode representar um erro sintático. Isto faz com que o processo de especificação, muitas vezes, torne-se parecido com o processo de implementação, onde a sintaxe é parte muito importante para a geração do código final. O usuário passa a se preocupar então com a sintaxe, quando toda a atenção deveria estar concentrada na especificação do sistema em si.

Isso leva a uma situação onde a especificação através de uma sintaxe textual muitas vezes é abandonada e parte-se então para o uso de gráficos desenhados em folhas de papel. Naturalmente, tem-se um quadro deturpado onde a TDF, que deveria servir como ferramenta de auxílio, obriga os usuários a usar métodos ultrapassados de definição.

Com o uso de sintaxes gráficas, entretanto, o desenvolvedor está livre de preocupações sintáticas, pois possui em suas mãos um conjunto de diagramas que constituem-se em uma ferramenta simples e intuitiva de especificação. A única preocupação envolvida é a especificação do sistema. Esta abordagem é mais eficiente pois todo o tempo dispensado é utilizado para a especificação, o que garante um melhor resultado final do trabalho realizado. Além disso, a correção sintática da especificação continua sendo garantida porque a conversão da sintaxe gráfica para a sintaxe textual deve obrigatoriamente ser possível.

No caso específico de LOTOS, tem-se ainda uma dificuldade adicional. A sintaxe textual possui uma abordagem matemática, o que foge completamente aos

conhecimentos que a maior parte dos usuários possuem. Em sua maioria, os profissionais que especificam os sistemas possuem um raciocínio fortemente relacionado às linguagens de programação, onde a ocorrência de eventos nos sistemas é dada a partir de uma abordagem seqüencial. O uso de LOTOS representa uma sensível mudança de paradigma para o usuário. Este fato muitas vezes faz com que os projetistas de software venham a preferir uma linguagem de especificação mais parecida com uma linguagem de programação.

A sintaxe gráfica tenta amenizar esta dificuldade inicial de compreensão de LOTOS por ser mais atraente ao usuário. Através de uma sintaxe gráfica é possível aproximar LOTOS dos conceitos aos quais os usuários estão mais familiarizados. Uma sintaxe gráfica pode, por exemplo, implementar diagramas de uma forma semelhante a fluxogramas. Apesar da sintaxe textual continuar apresentando uma abordagem matemática, a sintaxe gráfica permite a especificação de sistemas de uma forma mais intuitiva.

As proposta gráficas aqui apresentadas foram criadas procurando manter uma estreita correspondência com as construções LOTOS. Isto quer dizer que por mais atrativas que as mesmas venham a ser, a fidelidade com linguagem textual é mantida totalmente, tornando assim a conversão da sintaxe gráfica para a textual uma tarefa de tradução.

A seguir, apresentaremos duas das sintaxes gráficas para LOTOS: G-LOTOS e DART. Estas sintaxes serviram como base para a definição de uma nova sintaxe, chamada de E-DART, definida no próximo capítulo.

4.1 G-LOTOS

Uma especificação em G-LOTOS é composta tanto por elementos gráficos quanto textuais. Isso permite que os gráficos G-LOTOS sejam traduzidos para LOTOS de forma direta. Todos os diagramas são correspondentes diretos de construções LOTOS, onde tem-se um mapeamento um para um. Isto é, um gráficos G-LOTOS, quando traduzido, origina apenas uma construção LOTOS.

G-LOTOS é aplicável sobre todos os elementos de LOTOS, inclusive os tipos abstratos de dados. Aqui serão mostrados apenas os símbolos referentes aos operadores mais importantes da linguagem. Como estamos apenas interessados no comportamento das especificações e não como os dados são especificados, serão descritas apenas as construções que representam o comportamento dinâmico dos sistemas em G-LOTOS.

Processos em G-LOTOS

A figura a seguir apresenta o diagrama mais elementar de uma especificação em G-LOTOS: a definição de um processo. **ProcessName** define o nome do processo, *g1..gn* definem as portas de comunicação, enquanto *x1..xm* definem os parâmetros de entrada que são do tipo *t1..tm,* respectivamente. O principal elemento na definição dos processos é sua expressão de comportamento, representada pelo metasímbolo no formato de envelope nomeado por um *B*.

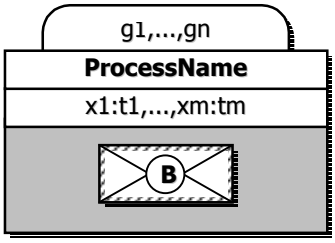


FIGURA 4.1 - Definição de processo

Sobre a expressão *B* podemos aplicar outras expressões de comportamento, definindo assim o comportamento do processo.

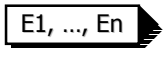
A seguir são apresentadas as expressões de comportamento aplicáveis sobre *B,* tanto em LOTOS como em G-LOTOS. Para a compreensão das mesmas devemos observar o seguinte:

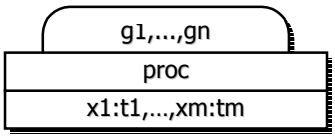
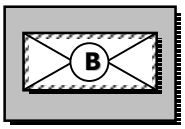
g, g1, ..., gn, a1, ..., am	identificadores de portas
x1, ...,xn, p1, ..., pm	variáveis
x: t	variável *x* do tipo *t*
E1, ..., En	expressões de comportamento

Expressões atômicas

A tabela a seguir apresenta as expressões de comportamento atômicas e sua representação gráfica em LOTOS.

TABELA 4.1 - Expressões atômicas

 `Stop`	***Inação.*** Representa um processo inativo que não pode interagir com seu ambiente.
 `exit`	***Terminação com sucesso.*** Sinaliza que o processo terminou com sucesso, e a próxima ação executada será a inação.
 `exit(E1,…,En)`	***Terminação com sucesso com passagem de parâmetros.*** Permite que sejam passados valores de retorna na finalização com sucesso de um processo.

 proc[g1,…,gn](p1,…,pm)	***Instanciação de processos.*** Indica a instanciação do processo *proc* com as portas *g1,...,gn* e parâmetros *x1,...,xm* dos respectivos tipo *t1,...,tm*.
 (B)	***Parênteses.*** Fornece o mecanismo para definição da ordem de avaliação das expressões de comportamento.

Expressões de composição paralela

As expressões de composição paralela de processos são modeladas através do uso do operador de comportamento paralelo. Tal operador apresenta-se de três forma diferentes na linguagem: o paralelismo de expressões que sincronizam-se através de uma lista de portas de sincronização, o paralelismo de expressões que sincronizam-se através de todas as suas portas de sincronização e o paralelismo de expressões que são avaliadas sem nenhuma sincronização. As duas últimas formas de paralelismo são na verdade casos particulares da primeira forma, visto que a lista de portas sobre as quais as expressões sincronizam-se pode ser vazia ou compreender todas as portas das expressões.

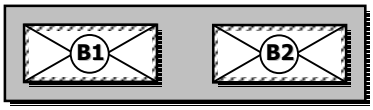
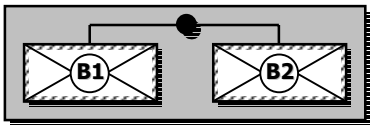
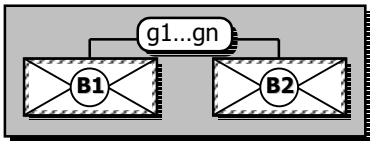
O paralelismo sobre um conjunto de portas é definido como sendo o resultado da composição em paralelo de duas expressões de comportamento com sincronização sobre um conjunto de portas que pode ser vazio ou compreender todas as portas das expressões.

O paralelismo de expressões de comportamento que não sincronizam-se é também chamado de paralelismo de processos independentes. A composição paralela de expressões é um caso particular da composição sobre uma lista de portas onde o operador de paralelismo é apresentado com um conjunto vazio de portas.

O paralelismo de expressões sobre todo o conjunto de portas é também conhecido como paralelismo de processos síncrono. Nesta forma de paralelismo, todas as ações das expressões de comportamento envolvidas devem ser realizadas de maneira síncrona em todas as portas. Esta composição é uma forma particular da composição paralela sobre uma lista de portas, onde tal lista compreende todas as portas das expressões envolvidas na sincronização.

A tabela a seguir apresenta as formas gráficas G-LOTOS para o operador de paralelismo e seus casos particulares.

TABELA 4.2 - Expressões de composição paralela

 B1 ||| B2	***Paralelismo de processos independentes.*** As expressões de comportamento *B1* e *B2* são avaliadas independentemente e em paralelo, sendo que entre as mesmas não ocorrerá nenhuma sincronização.
 B1 || B2	***Paralelismo do processos dependentes.*** As expressões de comportamento *B1* e *B2* serão avaliadas em paralelo, mas haverá sincronização entre elas em todas as portas.
 B1 |[g1…gn]| B2	***Paralelismo de processo generalizado.*** As expressões de comportamento *B1* e *B2* serão avaliadas em paralelo e ocorrerá sincronização apenas nas portas *g1,…,gn.*

Expressões prefixadas por uma ação

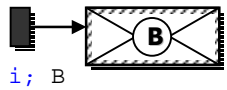
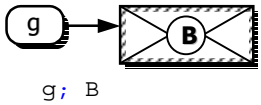
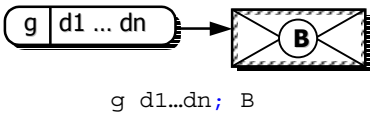
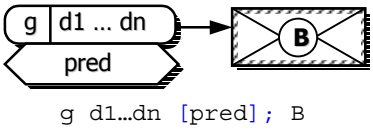
Existem quatro tipos de expressões de comportamento prefixadas por uma ação em G-LOTOS. A prefixação se caracteriza pela realização de uma ação antes que uma outra expressão de comportamento seja avaliada. Como uma expressão de comportamento em si também pode ser uma expressão prefixada, pode-se assim definir várias ações que serão avaliadas antes de uma expressão de comportamento em especial.

Uma ação de comportamento não observável é aquela que é interna a um processo e não pode ser observada fora do mesmo, o que impede esta ação não observável (ou ação interna) de interagir ou modificar o contexto do ambiente do processo. Por outro lado, uma ação observável pode interagir com o ambiente do processo, modificando seu contexto. Existem três tipos de ações observáveis: as ações básicas, as ações com passagem de valores e as ações com passagem de valores associadas a um predicado de seleção.

A ação observável básica não permite a troca de valores entre expressões de comportamento, mas promove a sincronização entre tais expressões. Para que haja a troca de valores entre as expressões envolvidas, deve-se utilizar as ações observáveis com passagem de valores. Esta passagem permite que dados possam ser enviados ou recebidos em uma sincronização. Por fim, a ação observável associada a um predicado de seleção é aquela que promove uma sincronização com troca de valores apenas se o predicado da ação for avaliado como verdadeiro.

A tabela a seguir apresenta as formas gráficas para as expressões de comportamento prefixadas por uma ação, bem como suas variações.

TABELA 4.3 - Expressões prefixadas por uma ação

	***Não observável.*** Representa a ação não observável *i* que é avaliada antes da expressão de comportamento *B*.
	***Observável.*** Representa a interação do processo com seu ambiente através da porta de comunicação *g*, antes da avaliação da expressão de comportamento *B*.
	***Observável com passagem de valor.*** Semelhante a expressão de comportamento observável, mas permitindo a passagem de valores em uma sincronização pela porta *g*.
	***Observável com passagem de valor e com predicado de seleção.*** Se *pred* resultar verdadeiro, então a sincronização com passagem de valor pela porta *g* pode ser avaliada.

Expressões adicionais

Como dito anteriormente, G-LOTOS possui construções para todos os elementos de LOTOS. Isso leva a um conjunto numeroso de diagramas. A seguir serão apresentados os outros diagramas que mais influenciaram no presente trabalho. Cabe ressaltar ainda que os diagramas não apresentados são igualmente importantes para a sintaxe G-LOTOS.

O operador de preempção permite a modelagem de um comportamento onde uma expressão pode interromper a avaliação de outra expressão. Pode-se, por exemplo, modelar com este operador políticas de ocupação de recursos em sistemas operacionais ou distribuídos.

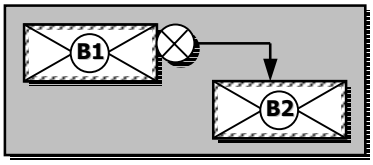
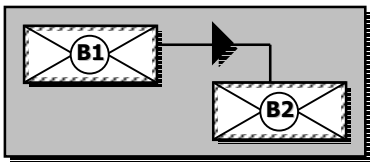
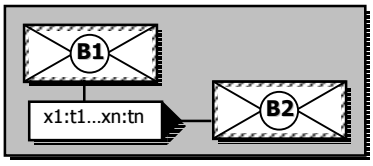
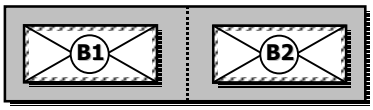
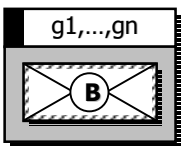
Enquanto o operador de preempção possibilitava a interrupção de uma execução por um outro processo, o operador de habilitação associa a avaliação de uma expressão de comportamento à finalização com sucesso de outra expressão. Assim, pode-se vincular a execução de uma expressão *B2* ao término com sucesso de uma expressão *B1*. O operador de habilitação, assim como as ações observáveis, pode apresentar troca de valores. Quando isso acontece, a expressão *habilitada* recebe valores da expressão avaliada anteriormente.

O operador de escolha permite a realização de uma seleção não determinística de expressões de comportamento. A expressão de comportamento, associada a um operador de escolha, será avaliada somente se esta expressão for a primeira a oferecer uma ação, o que *resolve* a seleção envolvida. Assim, para duas expressões *B1* e *B2* associadas a um operador de seleção, será executada aquela que primeiro realizar uma ação. Desta forma, dizemos que a expressão de comportamento em um operador de seleção é igual a expressão de comportamento que primeiro realizar uma ação.

Por fim, temos o operador *hide* que permite a ocultação de portas de sincronização, transformando ações observáveis de uma expressão de comportamento em ações internas. O operador define quais portas de sincronização de uma expressão de comportamento devem ser transformadas em ações internas. As portas ocultas pelo operador *hide* não poderão mais serem usadas para sincronizações externas, pois as mesmas agora são ações não observáveis.

A próxima tabela mostra a sintaxe gráfica para os operadores adicionais descritos.

TABELA 4.4 - Expressões adicionais

 `B1 [> B2`	***Preempção.*** A expressão de comportamento *B2* pode interromper a execução de *B1*, preemptando esta última e passando a ser avaliada.
 `B1 >> B2`	***Habilitação.*** Se a expressão de comportamento *B1* terminar com sucesso então a avaliação de *B2* estará habilitada e *B2* passará a ser executada.
 `B1 >> accept x1:t1…xn:tn in B2`	***Habilitação com passagem de valor.*** Se *B1* terminar com sucesso então *B2* será executada recebendo como parâmetros de entrada os valores *x1,...,tn*, dos respectivos tipos *t1,...,tn*, avaliados em *B1*.
 `B1 [] B2`	***Escolha.*** Dentre as expressões de comportamento *B1* e *B2* será avaliada apenas aquela que primeiro fornecer um comportamento ao ambiente.
 `hide g1,…,gn in B`	***Ocultação.*** Dentro da expressão de comportamento *B* as portas de sincronização *g1,...,gn* estarão ocultas, não podendo ser observadas externamente.

A expressão de comportamento *B* dos processos em G-LOTOS nada mais é que uma abstração de um comportamento. Analisando-se os gráficos apresentados

notamos que este tipo de construção permite a hierarquização das especificações, através de sucessivas abstrações de um comportamento.

Gerenciamento dos diagramas

G-LOTOS usa um tipo de construção onde novos elementos de uma especificação devem ser apresentados sempre dentro da área de seu predecessor. Analisemos o seguinte exemplo de instanciação de processo.

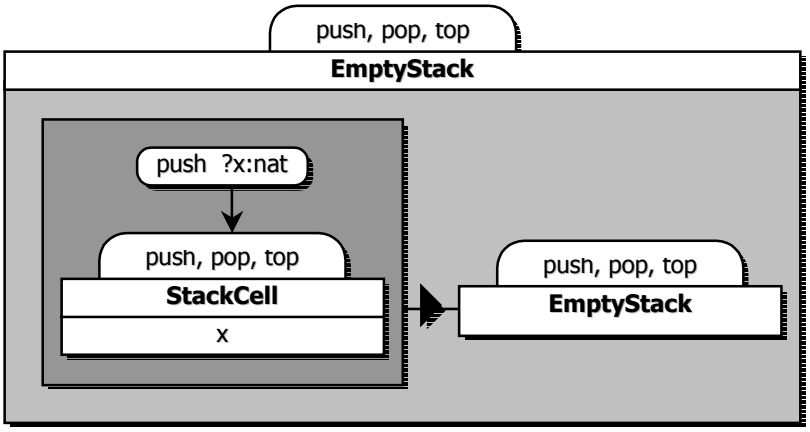


FIGURA 4.2 - Exemplo de instanciação de processo

Aqui, todo o comportamento do processo *EmptyStack* deve ser definido dentro dos limites do retângulo que representam o processo. A primeira ação executada é uma sincronização na porta *push* onde o valor resultante é associado à variável *x*. Após isso o processo *StackCell* é instanciado. Se esta primeira porção de ações terminar com sucesso, então a instanciação de um novo processo *EmptyStack* é permitida, fazendo com que o processo seja executado novamente.

Se quisermos colocar novos elementos isto deve ser feito sempre dentro do retângulo do processo. Por exemplo, digamos que sobre a variável *x* gostaríamos de aplicar alguma operação matemática antes da instanciação do processo *StackCell*. Isto deveria ser feito colocando-se novos gráficos entre o diagrama de sincronização e o diagrama de instanciação.

Para comportamentos com poucos elementos a linguagem G-LOTOS não apresenta problemas. Entretanto, nem todos os processos de um sistema possuem expressões de comportamento simples. Muitas vezes várias linhas de código LOTOS são necessárias para especificar-se tais processos. Nestes casos é que encontra-se uma limitação importante de G-LOTOS. Enquanto em um arquivo texto a inclusão de uma linha de código é algo trivial, em G-LOTOS a inclusão de um diagrama correspondente depende do espaço físico ainda disponível dentro do retângulo de um processo.

Expressões de comportamento muito grandes são extremamente difíceis de serem representadas em G-LOTOS porque o espaço disponível, na maioria das vezes, é insuficiente. Nestes casos, geralmente, a especificação deve ser fragmenta para que os gráficos não tornem-se inviáveis. Isto, entretanto, imprime ao usuário de G-

LOTOS a preocupação com o gerenciamento dos gráficos de uma especificação, gerenciamento este que com o uso da sintaxe textual não era necessário.

4.2 DART

DART (*Diagrams for Architectural Representation*) [ALL 97] é a contribuição da Universidade Politécnica de Madri para a solução do problema da definição de uma representação gráfica LOTOS. A abordagem apresentada em DART constitui-se na especificação de sistemas LOTOS através da definição *top-down* de processos e de relacionamentos entre estes processos.

A construção de DART foi baseada nos seguintes preceitos:

- Representar as interações de comunicação síncrona entre os processos de uma maneira clara;
- Ser uma notação gráfica para representar a arquitetura de um sistema;
- Permitir uma tradução para LOTOS;
- Permitir um conjunto de operações de alto nível, de forma a transformar a arquitetura dos sistemas de forma conveniente.

O sistema mais simples é representado por um único retângulo, chamado de DART, que representa um processo sem portas de comunicação e por conseguinte sem interações com outros processos.

Definição

Chamemos o retângulo do sistema acima de D. Assim, temos que o DART D é definido pela tupla (SD, G, SP, S, B) onde:

- SD é o conjunto de *sub-DARTs* pertencentes a D;
- G é o conjunto das *pontos de interação* pertencentes a D;
- SP é o conjunto de *pontos de sincronização* pertencentes a D. Um ponto de sincronização é definido como um conjunto de pontos de interação interconectados;
- S é o conjunto de *seqüências* pertencentes a D. Uma seqüência é definida como um par ordenado de DARTs;
- B é o conjunto de *desabilitações* (*preempções*) pertencentes a D. Uma desabilitação é definida como um par ordenado de DARTs.

Sub-DARTs

SD define o conjunto de sub-DARTs dentro do DART, isto é, SD define a composição do sistema como um todo. Cada sub-DART de SD define por sua vez o comportamento de uma parte do sistema. Isto permite que um DART possa ser definido recursivamente através de uma estrutura de outros DARTs. A figura a seguir mostra um sistema D que possui dois sub-DARTs (processos) D1 e D2.

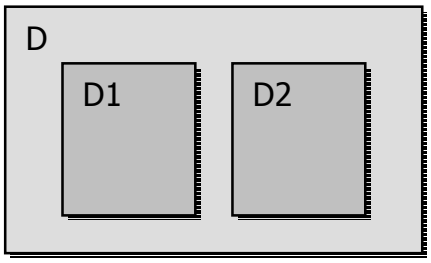


FIGURA 4.3 - DARTs e sub-DARTs

Pontos de Interação

G define o conjunto de pontos de interação de um DART com o seu ambiente. O sistema como um todo interage com outros DARTs através destes pontos, que são representados por um quadrado preenchido localizado nas bordas do retângulo que delimita o DART. Na figura a seguir podemos observar um DART D que possui três pontos de interação a, b, c.

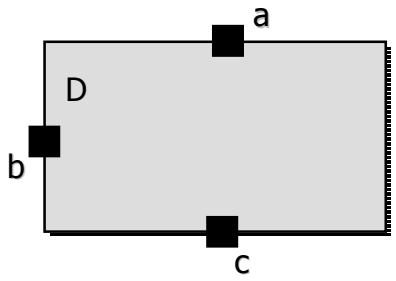


FIGURA 4.4 - Pontos de interação nos DARTs

Pontos de Sincronização

SP define o conjunto de pontos de sincronização. Se um ponto de sincronização é definido com três pontos de interação, apenas ações simultâneas sobre os três pontos de interação irão ocorrer. Pontos de sincronização são representados graficamente por círculos preenchidos que ligam os quadrados dos pontos de interação envolvidos na sincronização. Na figura a seguir podemos observar o DART

D que possui um ponto de sincronização para definir interações entre os pontos de interação dos sub-DARTs (processos) D1 e D2.

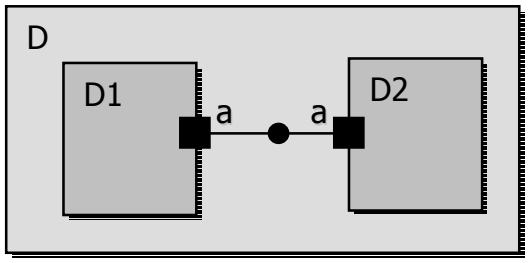


FIGURA 4.5 - Pontos de sincronização nos DARTs

Seqüência e Desabilitação

S define uma seqüência entre dois DART, D1 e D2, onde D2 começa a ser avaliado somente após D1 ter sido avaliado. S define assim as operações de habilitação de LOTOS. Já B define uma desabilitação (preempção) que um DART D2 pode executar sobre outro D1. Assim, D2 pode interromper D1, se este último estiver sendo avaliado. Neste caso D1 termina, e o controle é passado para D2, ocorrendo a preempção. Na figura a seguir podemos observar, à esquerda, o operador de seqüência, e à direita o operador de desabilitação, aplicados aos sub-DARTs D1 e D2.

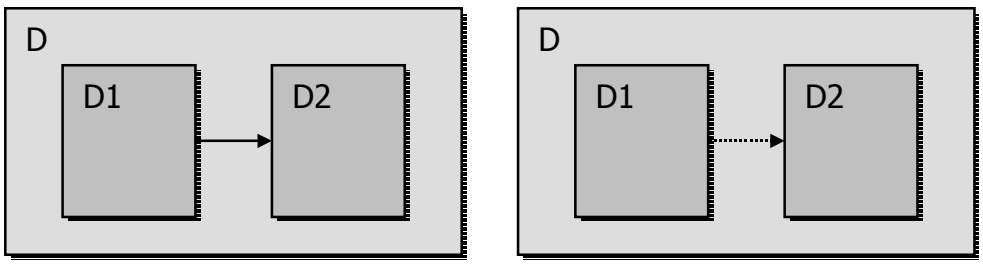


FIGURA 4.6 - Operadores de seqüência e desabilitação

Hierarquia de DARTs

Cada sub-DART pertencente a um DART é por si só outro DART que pode possuir também sub-DARTs. Assim, podemos criar especificações com vários níveis de abstração através da definição sucessiva de sub-DARTs. Cada novo conjunto de sub-DARTs define um novo nível de abstração. A figura a seguir apresenta um exemplo de um a hierarquia simples de DARTs.

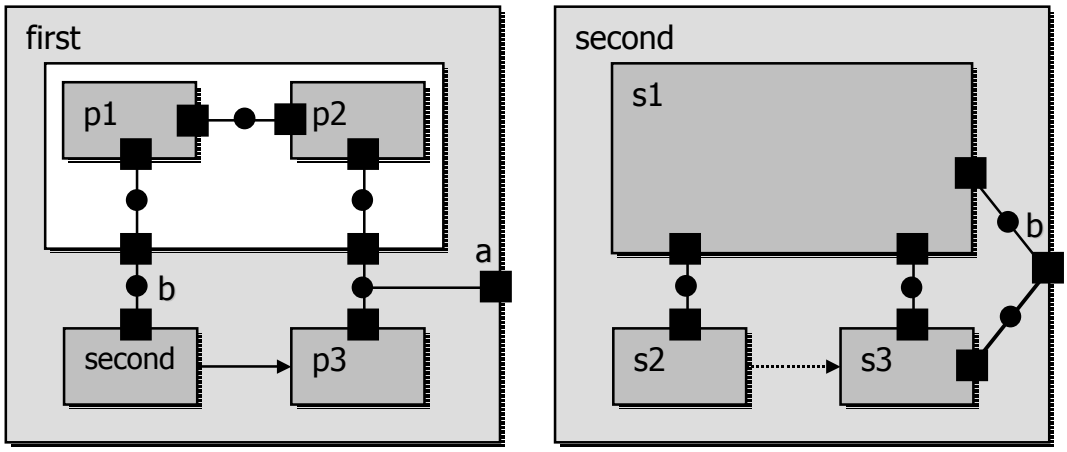


FIGURA 4.7 - Hierarquia de DARTs

Como podemos ver, *second* é associado a *first*. A correspondência entre os pontos de interação é dada pelo nome. Assim, é dito que o DART *second* é um refinamento de uma parte do DART *first*.

A proposta DART foi criada para representar a estrutura dos processos de um sistemas e suas relações. Deste ponto de vista, DART implementa esta funcionalidade de forma satisfatória. Entretanto, DART não representa estruturas de dados, visto que as observações importantes são em relação a arquitetura dos sistemas definidas em um alto grau de abstração.

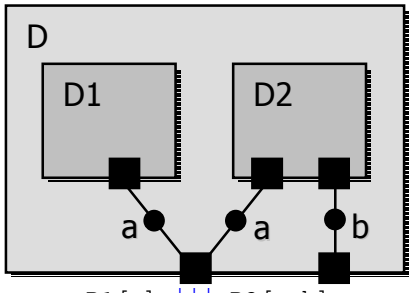
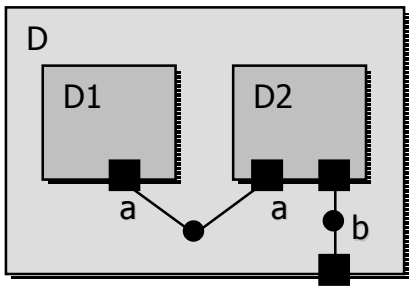
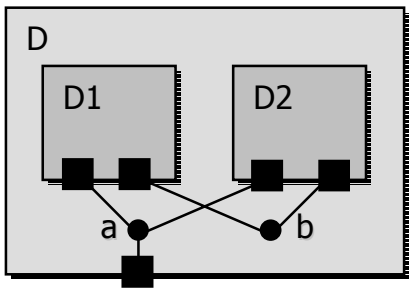
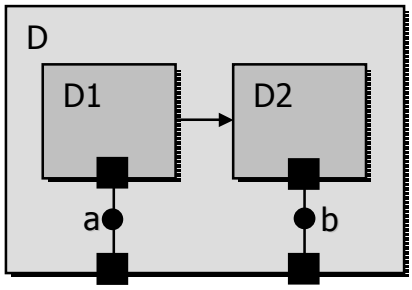
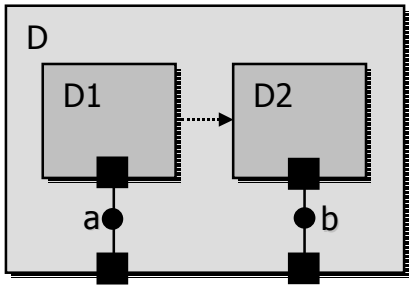
Mapeamento de DART para LOTOS

DART é capaz de representar o operador "hide", os operadores de paralelismo, a desabilitação, a seqüência e a hierarquia de processos. As outras construções LOTOS devem ser definidas por outra fonte de especificação, que pode ser a própria sintaxe textual a ainda uma segunda sintaxe gráfica.

Isso se deve ao fato de DART se preocupar apenas com a arquitetura dos sistemas, deixando de lado os comportamentos dos processos envolvidos. Assim, é definida uma função *lotos2dart* que, a partir de uma especificação textual LOTOS, chega-se a um diagrama hierárquico de DARTs. A função considera os operadores "hide" e de paralelismo (`||`, `|||` e `|[]|`) e a hierarquia de processos. Os outros operadores são transformados em DARTs elementares. Nota-se portanto, que uma transformação de DART para LOTOS sempre é possível. Entretanto o contrário não é verdadeiro, pois DART não suporta todas as construções LOTOS.

A figura a seguir mostra as correspondências entre os diagramas DART e a sintaxe textual de LOTOS.

TABELA 4.5 - Relações entre DART e LOTOS

 `D1[a] ||| D2[a,b]`	***Entrelaçamento.*** Os sub-DARTs (processos) *D1* e *D2* são avaliados independentemente pois não possuem sincronização entre si. *D1* interage com o ambiente externo (DART *D*) através do ponto de interação *a*, enquanto *D2* interage através de *a* e *b*.
 `hide a in D1[a] |[a]| D2[a,b]`	***Sincronização Parcial.*** *D1* e *D2* interagem através de seus respectivos pontos de interação *a*. O ponto de sincronização *a* denota tal interação. Como não há comunicação com o ambiente (DART *D*) o comportamento usa a construção "hide" para ocultar a sincronização.
 `hide b in D1[a,b] || D2[a,b]`	***Sincronização Completa.*** *D1* e *D2* sincronizam-se em todas as suas portas, o que denota uma sincronização completa. Novamente "hide" é usado para ocultar a sincronização em *b*, já que não existe sincronização com o ambiente através deste ponto.
 `D1[a] >> D2[b]`	***Habilitação.*** Se o sub-DART *D1*, que interage com o ambiente através de *a*, terminar sua execução com sucesso, então o sub-DART *D2*, que interage com o ambiente através de *b*, estará habilitado e será avaliado.
 `D1[a] [> D2[b]`	***Desabilitação.*** O sub-DART *D2*, que interage com o ambiente através de *b*, pode interromper o sub-DART *D1*, que interage com o ambiente através de *a*.

Operações Básicas Sobre DART

Sobre DART pode-se aplicar algumas operações para transformação dos diagramas. Tais operações estão divididas em operações estruturais: *Ungroup, Group, Flattering*, e transformações de comportamento: *Remove*. As principais operações são *Ungroup* e *Group*, pois possuem a propriedade de manter as relações entre os sub-DARTs. Isto significa que as conexões entre DARTs de níveis diferentes permanecem as mesmas independentemente do uso ou não de *Ungroup* e *Group*.

- *Ungroup*

A operação de *Ungroup* remove um nível de abstração de um DART. Alguns sub-DARTs pertencem a um mesmo nível de abstração do sistema. A aplicação da operação remove tal nível de abstração. O resultado é um DART onde as relações entre os sub-DARTs de níveis diferentes permanecem inalteradas.

- *Group*

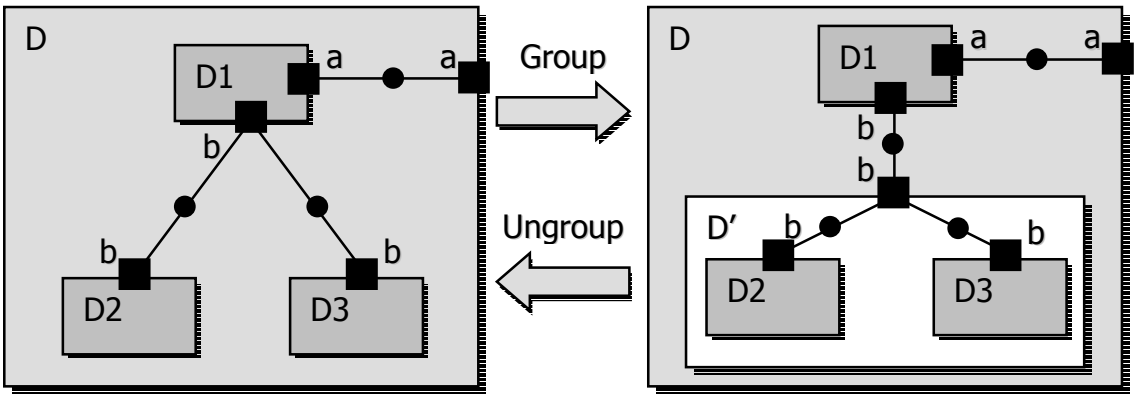
A operação de *Group* adiciona um nível de abstração a um sistema, agrupando um conjunto de sub-DARTs de um mesmo nível. Esta operação é útil na estruturação do sistema, onde os níveis de abstração podem ser criados sem afetar a funcionalidade existente. A próxima figura mostra um exemplo da aplicação das operações de *Ungroup* e *Group*.

- *Flattering*

A operação de *Flattering* transforma um DART com vários níveis de abstração em um DART com apenas um nível de abstração. Isso é feito aplicando-se sucessivas operações *Ungroup* sobre um DART. O diagrama a esquerda da figura a seguir é o DART resultante da aplicação de *Flattering* sobre o DART da direita.

- *Remove*

A operação de *Remove* remove a definição interna de um DART aumentando o nível de abstração. Entretanto há perda de informação. A operação é útil quando deseja-se uma melhor compreensão de partes do sistema sem que sua definição seja conhecida.




```
hide b in D1[a,b] |[b]| (D2[b] ||| D3[b])
```

FIGURA 4.8 - Operações *Group* e *Ungroup*

4.3 Comparação entre G-LOTOS e DART

G-LOTOS e DART são importantes extensões temporais porque, cada uma a sua maneira, apresentam alternativas gráficas para a substituição da sintaxe gráfica de LOTOS. A abordagem adotada pelas propostas são diferentes porque atacam aspectos diferentes da sintaxe textual. Desta forma, uma comparação entre G-LOTOS e DART pode levar a resultados diferentes, de acordo com as características analisadas. A comparação aqui apresentada é baseada na análise dos elementos mais importantes para se atingir os seguintes objetivos:

- A sintaxe gráfica deve ser a mais adequada e compatível com os conhecimentos do usuário;
- A sintaxe gráfica deve possuir os diagramas mais fáceis de serem adaptados ao E-LOTOS.

O primeiro objetivo diz respeito a adequação da sintaxe ao usuário e não do usuário à sintaxe. LOTOS não obteve grande sucesso no meio industrial devido à sua complexidade e estruturas matemáticas. Esta abordagem de especificação é totalmente nova para os usuários, acostumados a linguagens de programação procedurais ou orientadas a objetos. A forma de se *pensar* uma especificação em LOTOS é totalmente diferente do modo como os projetistas raciocinam. Quando estes usuários tentam manter um primeiro contato com a linguagem, esta é logo abandonada devido ao seu paradigma de especificação totalmente adverso. Parte-se então para o uso de métodos mais rudimentares de especificação ou então, no pior caso, o processo de especificação é deixado em segundo plano partindo-se direto para a implementação do sistema.

Diante deste quadro, o uso de uma sintaxe gráfica para LOTOS que não esteja adequada às necessidades do usuário está condenado ao insucesso. De nada adiante

uma sintaxe gráfica se esta não for utilizada. Temos então que o simples uso de gráficos não resolve o problema que acontecia com a sintaxe textual. O importante aqui não são os gráficos em si mas o modo como o usuário manipula os mesmo para construir suas especificações.

O segundo objetivo apontado diz respeito especificamente a E-LOTOS. Estamos interessados em linguagens gráficas particularmente fáceis de serem alteradas para que possam serem utilizadas também na especificação de sistema através de E-LOTOS. Desta forma, a sintaxe gráfica deve inevitavelmente possuir construções para o tratamento explícito da noção de tempo quantitativo.

Assim, estamos interessados em uma linguagem *facilmente expansível* em termos de suas construções gráficas. A inclusão de novos elementos não deve afetar a forma como os elementos já existentes são apresentados, permitindo assim uma certa compatibilidade entre especificações criadas com a sintaxe gráfica original e novas especificações construídas com a sintaxe ampliada.

Tendo em vista os dois objetivos apresentados, seremos capazes de identificar os pontos mais fortes das sintaxes gráficas DART e G-LOTOS. Isto permitirá a criação de uma nova sintaxe gráfica, apresentada no próximo capítulo, que será a composição dos aspectos mais importantes de DART e G-LOTOS aplicados a E-LOTOS.

Adequação aos conhecimentos do usuário

Como dito na apresentação de G-LOTOS, esta sintaxe gráfica é baseada no mapeamento direto das estruturas textuais de LOTOS para um correspondente gráfico. A grande vantagem existente neste fato é a capacidade que G-LOTOS tem de poder representar qualquer sistema especificado em LOTOS. Como o mapeamento é direto, qualquer estrutura textual pode ser transformada em uma estrutura gráfica, assegurando a total compatibilidade entre G-LOTOS e LOTOS. O processo de tradução de G-LOTOS para LOTOS e vice-versa é trivial pois envolve apenas a troca de um gráfico por um texto ou de um texto por um gráfico.

Mas apesar desta vantagem apresentada por G-LOTOS, a principal característica existente no mapeamento direto das estruturas textuais para gráficos está no fato de que os tipos de relações entre estruturas textuais também são encontrados nas estruturas gráficas. Isto é, o *paradigma* das especificações gráficas é igual ao encontrado nas especificações textuais. Tendo em mente que o paradigma existente na sintaxe textual não é adequado aos conhecimentos do usuário, concluímos que o modo como os gráficos são apresentados por G-LOTOS também não é adequado. Pode-se dizer que as complexidades textuais sentidas pelos usuários foram todas transferidas para a sintaxe gráfica.

Este fato inviabiliza a sintaxe gráfica enquanto solução para o problema da necessidade de simplificação das estruturas textuais de LOTOS. O usuário em contato com G-LOTOS terá as mesmas dificuldades de compreensão que encontrava em LOTOS. G-LOTOS, sobre este ponto de vista não apresenta nenhuma melhoria em relação a LOTOS.

Por outro lado, DART, por apresentar uma abordagem diferente na construção das especificações, mostra-se mais interessante aos olhos do usuário. Como mostrado

na apresentação da sintaxe gráfica, as especificações são construídas enfocando-se primeiramente a definição dos processos envolvidos e a comunicação existente entre os mesmos. Só então o comportamento dos mesmos é definido. Para o usuário é mais natural *pensar* um sistema com processos sendo executados em paralelo e que comunicam-se através da troca de mensagens que trafegam em canais definidos. Esta abordagem é mais simples que, por exemplo, imaginar que um sistema é constituídos de várias expressões gráficas/matemáticas que definem seu comportamento, como ocorre em G-LOTOS.

Como os gráficos em DART são extremamente simples a compreensão do sistema é melhorada mais ainda. A abstração de processos que são executados dentro de outros processos também é facilitada pelos diagramas, o que consequentemente ajuda na criação de sistemas, principalmente quando especificamos tais sistemas de forma *top-down*. Para exemplificar o poder de expressão alcançado com DART temos dois exemplos de construções gráficas para a seguinte sentença LOTOS:

```
D = hide a in D1[a,b] |[a,b]| D2[a,b]
```

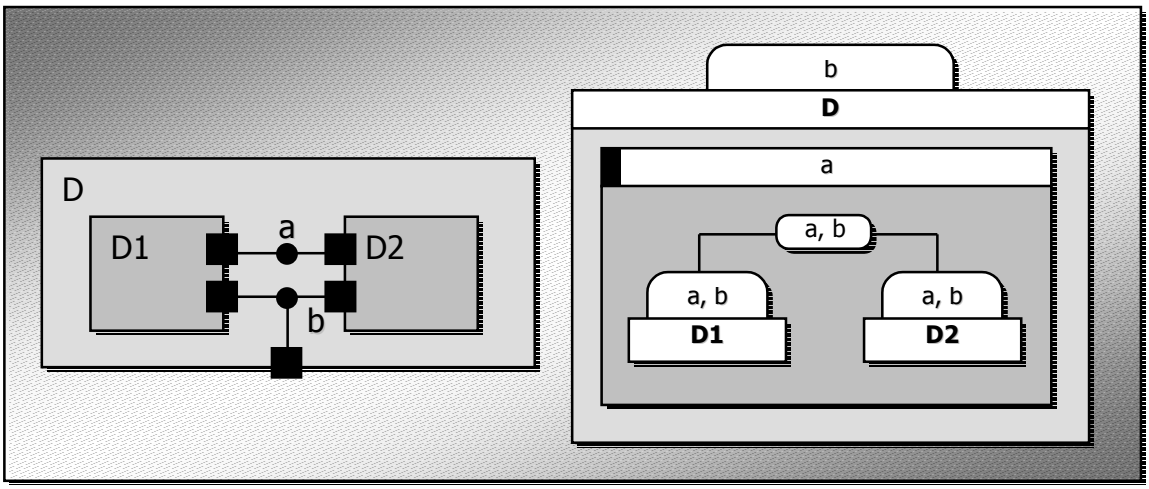


FIGURA 4.9 - Exemplo DART e G-LOTOS

A diferença de complexidade e expressividade dos dois exemplos é evidente. Enquanto em DART necessitamos de poucos retângulos para especificar os processos e suas portas, em G-LOTOS devemos utilizar de várias construções para alcançar o mesmo resultado. O exemplo evidencia também o nível de abstração, em relação à sintaxe gráfica, alcançado. Em DART as ligações realizadas entre as portas de comunicação por si só já definem onde será empregado o operador “hide”. Esta definição é bem mais intuitiva que em G-LOTOS, onde, como podemos notar no diagrama, deve-se indicar o “hide” de forma explícita. Além de DART ser mais intuitivo, seus diagramas requerem poucos símbolos para expressar convenientemente operações complexas.

Entretanto, diferentemente de G-LOTOS, os diagramas DART não são mapeamentos diretos de estruturas textuais. Isto significa que um único diagrama pode ser traduzido em várias construções textuais complexas. Assim, a tradução de DART para LOTOS não é um processo trivial, muitas vezes exigindo um intenso processamento sobre os gráficos envolvidos. Outra conseqüência disso é que podem haver inconsistências estruturais nos gráficos DART que são apenas notadas quando

tentamos converter os gráficos para LOTOS. Assim, apesar dos gráficos DART serem mais adequados aos conhecimentos prévios dos usuários, estes gráficos devem ser usados com cuidado para garantir a integridade formal das especificações.

Facilidade de adaptação da linguagem a E-LOTOS

Apesar de DART ser a mais adequada aos conhecimentos prévios do usuário, como vimos anteriormente, a mesma não é de fácil adaptação a E-LOTOS. O grande poder de expressão de DART está calcado na clara definição dos processos de um sistema, suas portas de sincronização e a forma como estes processos interagem. Em um alto nível de abstração esta funcionalidade é muito importante para se ter uma visão mais abrangente dos sistemas. Entretanto, a medida em que vamos refinando as especificações surge a necessidade da definição do comportamento interno dos processos. Neste ponto, devido a inexistência de outras construções DART, a tarefa de especificação torna-se mais complicada. Como apresentado na introdução da linguagem, DART é capaz de expressar apenas as várias formas do operador de paralelismo, o operador de habilitação e o operador de preempção.

A definição de novos operadores para DART, inevitavelmente, tornaria a linguagem mais complexa nos níveis onde a definição dos comportamentos internos dos processos se faz necessária. Desta forma, a adaptação de DART a E-LOTOS não é algo fácil e introduz inevitáveis complexidades.

Já em G-LOTOS, a definição do comportamento interno dos processos é uma tarefa mais factível, novamente, pela existência de diagramas gráficos para todas as construções textuais da linguagem. Como dito anteriormente este fato aumente drasticamente a complexidade dos gráficos, mas pode-se com isso definir qualquer sistema, tanto em níveis mais abstratos quanto ao nível da definição do comportamento interno dos processos. Tendo em vista a complexidade intrínseca da linguagem, a inclusão de novos operadores não introduz novas complexidades. Facilmente pode-se criar novas estruturas gráficas que correspondem às construções específicas introduzidas em E-LOTOS. Assim, em relação a adaptação a E-LOTOS, G-LOTOS mostra-se mais adequada que DART, devido a forma como os diagramas são apresentados e pelo fato de poder-se especificar o sistema em seus vários níveis de abstração.

Outros aspectos de DART e G-LOTOS

Apesar das principais comparações entre DART e G-LOTOS girarem em torno dos dois objetivos apresentados anteriormente, existem outros aspectos destas linguagem que devem ser levados em conta, em maior ou menor escala, em uma comparação.

Em relação à G-LOTOS um aspecto importante é a dificuldade de criação de sistemas complexos levando em conta a forma como os gráficos são organizados. Isso retoma o problema, antes citado na apresentação de G-LOTOS, do acúmulo de gráficos dentro de um mesmo retângulo. Este problema aumenta ainda mais a complexidade dos gráficos existentes. Se antes a complexidade era devido ao mapeamento direto das

estruturas textuais, agora temos ainda o fato de que os gráficos em si serem de difícil visualização.

Já em relação a DART, um problema importante é a já citada incapacidade da linguagem de expressar convenientemente o comportamento interno dos processos. A falta de símbolos para todas as estruturas textuais faz de DART uma linguagem simples mas incompleta. Os diagramas apresentados são claros e fáceis de serem entendidos, mas quando há a necessidade de detalhamento DART mostra-se inadequado.

Isso se deve ao fato de que DART foi concebido para expressar apenas os níveis mais abstratos de uma especificação, no qual é muito competente. Segundo sua concepção, os níveis mais detalhados deveriam ser especificados textualmente ou até mesmo com o uso de G-LOTOS. Na verdade DART foi concebido como uma ferramenta de auxílio nos primeiros passos de uma especificação. Este fato pode ser observado da descrição da ferramenta GLD (*GraphicaL Designer*) desenvolvida também pela Politécnica de Madri. A ferramenta suporta DART nos níveis mais abstratos e G-LOTOS nos níveis mais refinados.

DART ou G-LOTOS?

Apesar de G-LOTOS mapear para uma sintaxe gráfica todos os elementos de LOTOS (inclusive os referentes aos tipos abstratos de dados), G-LOTOS não soluciona de forma satisfatória o problema da complexidade. Um diagrama em G-LOTOS é de difícil compreensão mesmo para quem tem familiaridade com a linguagem. O ponto crucial desta complexidade reside no fato de que, por G-LOTOS fazer uma correspondência um para um com LOTOS, G-LOTOS não descreve claramente os sistemas. Soma-se a isso a falta de clareza gerada pelo acúmulo de gráficos dentro dos retângulos que formam os diagramas.

Por outro lado, DART apresenta uma proposta bem mais adequada a representação dos sistemas. Os diagramas DART são de fácil entendimento porque os principais elementos são representados de forma bem mais clara que em G-LOTOS. É trivial, por exemplo, identificar os pontos de interação entre processos e a forma como estes processos interagem através dos pontos de sincronização. Entretanto, o principal problema de DART é que a sintaxe gráfica não apresenta símbolos necessários para se representar todas as construções LOTOS. O operador de escolha, por exemplo, não possui uma representação gráfica, deixando a desejar o poder de expressão da linguagem quando se deseja especificar mais detalhadamente um sistema.

Em relação aos objetivos que procuramos alcançar com o uso de uma sintaxe gráfica, temos que DART é mais adequado tendo em vista os conhecimentos dos usuários não familiarizados com E-LOTOS. DART representa eficientemente os sistemas de uma forma que as complexidades inerentes à sintaxe textual são abstraídas na linguagem gráfica. Por outro lado, G-LOTOS mostrou-se de mais fácil adaptação a E-LOTOS. Novas construções podem ser criadas sem alterar a forma como a linguagem original funciona. Por outro lado, G-LOTOS mostrou-se muito complexo para ser efetivamente usado. Suas complexidades são tão grandes, ou até mesmo maiores, que as complexidades da linguagem textual.

Por fim, chega-se a conclusão quem nem G-LOTOS nem DART apresentam uma sintaxe gráfica suficientemente interessante ou completa para expressar uma especificação textual LOTOS. DART é muito claro e atende perfeitamente ao requisito de simplicidade, mas não é suficientemente completo. G-LOTOS é tão completo quanto o próprio LOTOS, mas não é capaz de expressar de forma simples uma especificação textual LOTOS.

A resposta para a pergunta “DART ou G-LOTOS?” é então nenhum dos dois. A solução ideal para o problema é ter-se uma sintaxe gráfica que combine as melhores características de G-LOTOS e DART. Uma sintaxe ampla, sem deixar de ser simples, capaz de expressar os sistemas E-LOTOS em seus vários níveis de abstração de forma eficiente mas intuitiva. O próximo capítulo apresenta E-DART, a nova sintaxe gráfica proposta neste trabalho, que procura alcançar estes objetivos.

5 A Sintaxe Gráfica E-DART Proposta

Como vimos no capítulo anterior, G-LOTOS e DART são importantes extensões gráficas de LOTOS, mas que não satisfazem completamente os requisitos necessários para sua efetiva utilização. Apesar disto, cada uma das linguagens possui características importantes que podem basear a criação de uma nova sintaxe gráfica mais adequada às nossas necessidades.

Assim, neste capítulo é apresentado E-DART (*Enhancements to DART*), uma nova sintaxe gráfica baseada em G-LOTOS, mas principalmente em DART. Por apresentar, de forma geral, melhorias à DART, e ter como objetivo principal a representação de sistemas E-LOTOS, a abreviação de *Enhancements to DART* mostrou-se um nome coerente para a sintaxe.

A figura a seguir apresenta um gráfico representativo do trabalho realizado. Baseados em E-LOTOS, DART e G-LOTOS, temos como resultado a criação da sintaxe textual E-DART e da ferramenta Editor E-DART, apresentada no próximo capítulo.

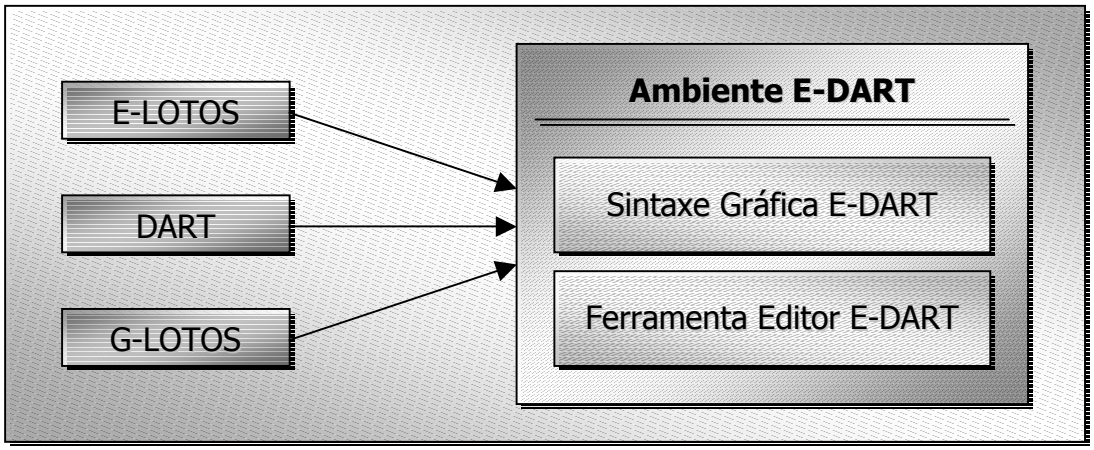


FIGURA 5.1 - O ambiente E-DART e sua construção

O ponto de partida da definição de E-DART leva em conta que o processo de especificação de sistemas, na maior parte dos casos, é feito de uma maneira *top-down*. Iniciando com a especificação dos níveis mais altos de abstração, aos poucos vai-se refinando o sistemas através de níveis de abstração cada vez mais próximos da implementação. Tendo em vista este fato, e a análise apresentada sobre DART no capítulo anterior, chega-se a conclusão que as primeiras definições de um sistema podem ser adequadamente construídas com os diagramas DART padrão.

Assim, E-DART, quando da especificação dos níveis mais abstratos de um sistema, é praticamente idêntico a DART. As diferenças entre as linguagem começam a aparecer quando tem-se o refinamento dos sistemas. Enquanto DART é incapaz de representar o comportamento neste nível de especificação, E-DART deve possuir mecanismos para tal. Neste ponto E-DART recebe uma forte influência de G-LOTOS, que é capaz de representar qualquer nível de abstração de um sistema.

5.1 A Construção de E-DART

A construção de E-DART foi orientada segundo dois princípios básicos:

- Minimizar ao máximo as complexidades inevitavelmente inseridas em DART;
- Apresentar uma forma de especificação adequada aos conhecimentos dos usuários.

Estes dois princípios estão de acordo com os objetivos que procuramos alcançar em uma sintaxe gráfica para LOTOS, descritos no capítulo anterior.

Minimizar ao máximo as complexidades inseridas em DART significa dizer que, em primeiro lugar, complexidades serão necessárias para que se possa representar os níveis de abstração mais próximos da implementação. Entretanto, estas complexidades devem causar o menor impacto possível no entendimento dos sistemas como um todo.

Apresentar uma forma de especificação adequada aos conhecimentos dos usuários refere-se ao fato de que as construções inseridas em DART devem manter a facilidade que o próprio DART proporcionava de compreensão dos sistemas. Deve-se manter um padrão de entendimento das especificações. Como DART mostrou-se adequado aos conhecimentos prévios dos usuários na hora de se especificar sistemas, o fato do padrão de compreensão das especificações ser mantido permitirá que E-DART seja tão adequado quanto DART aos conhecimentos dos usuários.

O ponto de partida inicial para a construção de E-DART, naturalmente, foram os diagramas DART. A partir destes diagramas tentou-se inserir novas construções que permitissem a definição de comportamentos antes não representáveis em DART. As construções escolhidas inicialmente foram as seguintes:

- Seqüência
- Escolha
- Ação Interna
- Sincronização
- Stop
- Exit

Com isto uma primeira abordagem de E-DART foi criada para suportar estes operadores. Para demonstrar esta abordagem tomemos o exemplo apresentado no capítulo 2 de um sistema para teste de uma ambiente de comunicação. A sistema é especificado de acordo com as seguintes construções LOTOS.

```
TESTER := send;(receive;success;exit [] i;TESTER)
MEDIUM := send;(receive;MEDIUM [] i;MEDIUM)
hide send,receive in TESTER |[send,receive]| MEDIUM
```

Para este sistema temos o seguinte diagrama (fig 5.2).

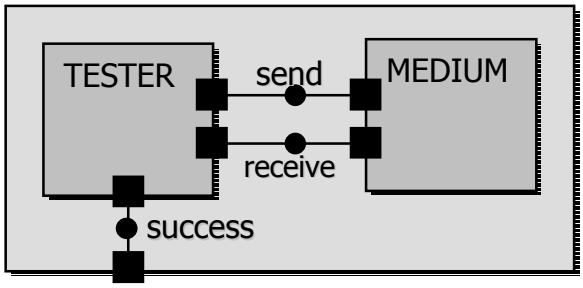


FIGURA 5.2 - Sistema para teste de ambiente em E-DART

Neste diagrama é apresentado o nível de abstração mais alto do sistema. Portanto, os diagramas presentes são totalmente idênticos à DART e nenhuma nova construção, neste nível, foi acrescentada à sintaxe gráfica. Neste caso específico, temos os dois processos envolvidos no sistema definidos por dois retângulos. Cada processo possui duas portas (*send* e *receive*) usadas para a troca de informações. O processo *TESTER* possui ainda uma porta *success* para indicar o sucesso de um teste. Os pontos de sincronização indicam a comunicação ente os processo, além de especificarem, de forma indireta, o operador "hide" de LOTOS. Isto é feito porque os pontos de sincronização que não são ligados a nenhuma porta do ambiente devem ser *escondidos* de interações externas.

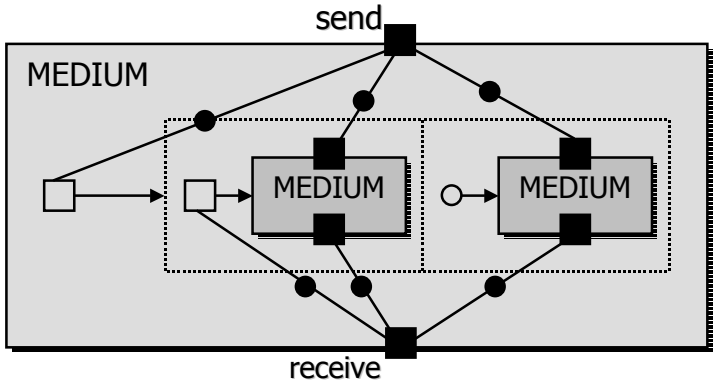


FIGURA 5.3 - Processo *MEDIUM*

A figura 5.3 apresenta a versão gráfica para o processo *MEDIUM*. Neste nível de abstração, onde o comportamento de um processo precisa ser especificado, encontramos as primeiras modificações introduzidas por E-DART. A primeira delas refere-se à ação de sincronização, denotada por um retângulo similar ao retângulo identificador das portas de sincronização. A única diferença é o fato de que em uma sincronização o retângulo não encontra-se preenchido. Devemos notar ainda que o retângulo de sincronização está ligada à porta de sincronização correspondente, indicando em que porta a sincronização está acontecendo. Assim, em *MEDIUM*, temos uma primeira sincronização na porta *send*.

A segunda nova construção encontrada é o operador de seqüência, denotado por uma linha comum com uma seta em seu destino. Isso permite identificar a ordem de avaliação das ações em um comportamento. No processo *MEDIUM,* temos então que após a primeira sincronização, as ações internas ao retângulo pontilhado serão avaliadas. Este retângulo pontilhado, por sua vez, define a próxima construção incorporada à linguagem. Trata-se do operador de escolha. O operador define quais ações podem ser avaliadas dentro do retângulo pontilhado. As ações possíveis são delimitadas internamente ao retângulo por áreas distintas. No caso de *MEDIUM* temos duas possibilidades de escolha.

A primeira possibilidade de escolha é trivial e denota uma sincronização na porta *receive* e uma nova instanciação do processo *MEDIUM.* Já na segunda opção temos a introdução de outra nova construção. O círculo não preenchido que precede uma nova instanciação de *MEDIUM* representa o operador de ação interna ***i***, que no caso de *MEDIUM* especifica a perda da mensagem inicial, modelando uma falha no ambiente de comunicação.

Observemos agora a definição gráfica do processo *TESTER*, que encerra a especificação do sistema de testes de ambientes de comunicação.

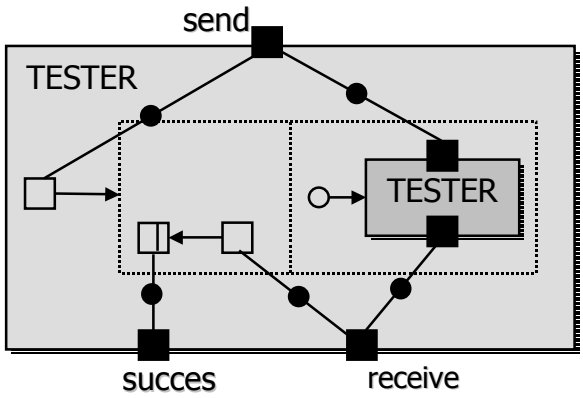


FIGURA 5.4 - Processo *TESTER*

Em *TESTER* podemos observar a presença de mais um novo operador gráfico. A sincronização final em *success* indica o sucesso no teste de confiabilidade do ambiente de comunicação. Após esta indicação o processo *TESTER* deve ser terminado com sucesso. Isso é indicado pelo retângulo vazado ao lado da sincronização em *success.* Assim, este retângulo corresponde ao operador *exit* da sintaxe textual de LOTOS. Por fim, para o operador final *stop* tem-se uma construção similar a *exit*, apenas tendo como diferença o fato de que o retângulo usado para indicar um *stop* é preenchido internamente.

Este exemplo é útil para identificarmos os pontos onde a sintaxe gráfica pode ocorrer em erros e ajuda a definir uma forma mais apropriada para as definições gráficas. A primeira abordagem de E-DART apresentada é baseada na criação de novos operadores *internos* aos diagramas que limitam os processos DART. Como conseqüência imediata desde fato temos os problemas apresentados em G-LOTOS para complexidade dos gráficos. Como todos os elementos de um processo devem estar internos ao seu retângulo, processo complexos são muito difíceis de serem especificados graficamente. Outro problema decorrente de definições *internas* aos processos é o número excessivo de linha nos diagramas. Como para cada

sincronização é necessária uma linha que identifica a qual porta essa sincronização está associada, para processos com muitas sincronização o número de linha existentes pode ser muito grande, inviabilizando a legibilidade dos gráficos.

A grande vantagem encontrada em DART é o fato do mesmo especificar claramente os processos dos sistemas a forma como esses processos interagem. E-DART tem por objetivo incluir a capacidade de se especificar o comportamento interno de cada um desses processos. Esta primeira abordagem tenta fazer isto incluindo gráficos novos internamente aos limites de cada processo, mas isso, como comprovado anteriormente, aumenta a complexidade dos gráficos, anulando a clareza inicial apresentada por DART. A solução para o problema está em incluir novas construções *externamente* aos processos, isto é, cada novo operador deve estar contido fora dos limites dos processos. As sessões a seguir mostram como isso é alcançado e as modificações em DART necessárias para tornar isso possível.

5.2 Restringindo DART: Tornando a Linguagem Consistente

Como dito anteriormente, a grande vantagem de DART está no fato de sua sintaxe definir de forma clara os processos de um sistema e a forma como os mesmos interagem. DART, por outro lado não possui a capacidade de definir o comportamento interno de cada um destes processos. Os gráficos DART permitem ver os sistemas de um nível de abstração muito alto. A medida que queremos um refinamento das especificações, DART apenas é capaz de indicar os processos envolvidos nesses refinamentos, mas não seu comportamento.

Tomando esses fatos como base, E-DART é construído para permitir a definição interna dos comportamentos dos processos. E-DART deve expandir DART para que essa definição seja possível.

Entretanto, existem duas exceções nos diagramas em DART que são importante para o resultado final de E-DART: o operador de preempção e o operador de habilitação. Estes operadores se constituem em exceção porque eles efetivamente definem um comportamento para os processos onde são empregados. Além disso, estes operadores encontram-se internos aos limites dos processos, fato que, como vimos anteriormente, leva a gráficos ilegíveis para sistemas com processos complexos.

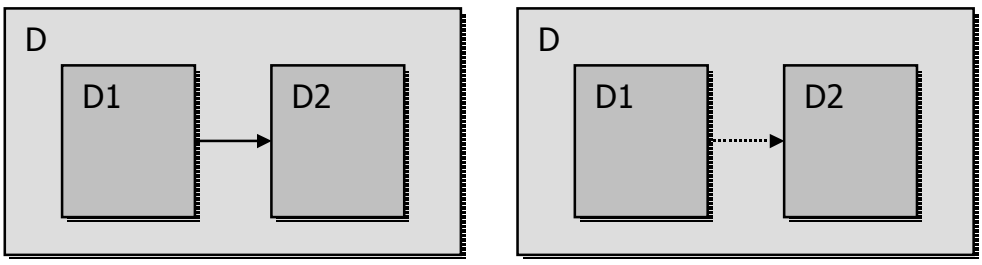


FIGURA 5.5 - Operadores de seqüência e desabilitação

A existência de uma construção gráfica para a preempção e habilitação torna as futuras construções E-DART inconsistentes com o resto da sintaxe gráfica. A solução mais natural para o problema seria a redefinição destas construções baseadas nas novas construções E-DART. Isto é, eliminaríamos a forma como a preempção e a habilitação são representadas graficamente e novas construções para estes operadores seriam criadas.

Devemos ainda levar em conta um outro fato. A sintaxe textual E-DART deve ser voltada para a definição de sistemas baseados em E-LOTOS, e não LOTOS. Num primeiro momento, este fato não possui grande importância, mas devemos considerar o fato de que o operador de habilitação e o operador de seqüência, em E-LOTOS, foram substituídos por um novo operador de seqüência. Isso leva a um resultado que vêm a nos auxiliar na consistência de E-DART. O operador de habilitação deve ser excluído de DART não apenas para permitir uma linguagem coerente, mas também porque o mesmo não existe mais em E-LOTOS. Assim, E-DART não deverá propor um operador de habilitação substituto, apenas sendo necessária a criação de uma nova estrutura para o operador de preempção.

5.3 Relação entre Gráficos e Apresentação de Informação

A utilização de gráficos como uma alternativa à sintaxe textual é necessária para a abstração de complexidades que não são importantes no processo de especificação dos sistemas. O processo de abstração envolve a supressão de informações que julgamos não serem importantes em um determinado momento. É justamente essa supressão de informação que permite a simplificação dos gráficos e os tornam mais claros que a sintaxe textual. Quanto mais informação for suprimida, mais abstratos se tornam os gráficos existentes. Quanto mais informações são apresentadas, mais complexos são estes gráficos.

Os fatores que determinam quão complexa é uma linguagem gráfica são muitos. Já analisamos, por exemplo, a forma como os gráficos são apresentados, a dificuldade de adaptação destes gráficos aos conhecimentos prévios dos usuários, entre outros. O conjunto de informações que uma sintaxe gráfica disponibiliza ao usuário também faz parte deste conjunto de fatores que tornam a linguagem mais ou menos complexa.

Para exemplificar este fato, tomemos novamente DART e G-LOTOS. Façamos uma comparação entre dois diagramas usados para a definição de um processo, sem nos preocuparmos por enquanto com o seu comportamento interno.

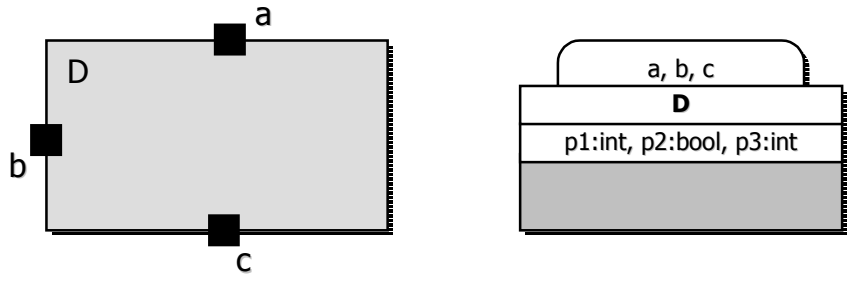


FIGURA 5.6 - Definição de um processo em DART e G-LOTOS

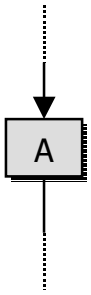
No exemplo da figura acima temos a definição de um processo *D* que possui as portas de sincronização *a, b* e *c* e parâmetros de instanciação *p1, p2* e *p3* cujos tipos são *int, bool* e *int*, respectivamente. Como já mostrado anteriormente, os gráficos DART apresentam as informações de uma forma mais clara que G-LOTOS. Isso pode novamente ser confirmado pelo exemplo. Entretanto, em DART não é possível saber quais os parâmetros de instanciação do processo, como acontece em G-LOTOS, onde esta informação é definida explicitamente através de uma lista de parâmetros e tipos associados.

O fato de DART por si só ser mais claro que G-LOTOS, entretanto, não significa que a presença dos parâmetros de instanciação não seja importante. A ausência de tais parâmetros nos gráficos simplificam os mesmos, e torna a observação do sistema mais simples. Mas quando o usuário está interessado em um processo em particular e não no sistema como um todo, a ausência de parâmetros pode representar uma dificuldade de compreensão para este processo em particular.

Em suma, uma informação é relevante de acordo com o modo em que as informações correlatas são analisadas pelo usuário. A importância de uma informação depende então do ponto de vista do observador. Os parâmetros de um processo podem não ser importantes se estamos interessados no sistema como um todo. Entretanto, se observamos um processo independentemente do sistema, os parâmetros de instanciação são informações importantes.

Tendo estes fatos em vista chegamos a conclusão que as todas as informações de um sistema são importantes. Tudo depende de como observamos as especificações. Assim, não é conveniente descartar qualquer informação, pois corre-se o risco de invalidar alguma etapa do processo de especificação. Neste caso, o desafio é encontrar uma forma de disponibilizar todas as informações sem prejudicar a clareza das especificações.

Ocorre-se um erro se tentarmos usar gráficos para todos os dados. A complexidade aumentaria e teríamos o mesmo efeito alcançado em G-LOTOS. O gráficos devem continuar sendo simples e claros. As outras informações devem ser apresentadas por formas alternativas. E uma forma alternativa eficiente de se disponibilizar as informações é o emprego de uma tabela de propriedade para cada um dos elementos gráficos. Esta tabela possui todos as propriedades complementares para que um elemento gráfico possa ser plenamente especificado. Tomemos como exemplo a seguinte figura.



Parâmetros	?x,!2
Tempo	t
Condição	t<4
Início	2

FIGURA 5.7 - Exemplo de uma tabela de propriedades

O gráfico representa um processo que executa uma ação de sincronização na porta *A*. De acordo com E-LOTOS, uma sincronização, na sintaxe textual, é definida pelo seguinte (ver capítulo 3).

```
G[P1][@P2][[E]][start(N)]
```

G é a porta de sincronização usada, *P1* são os parâmetros trocados na sincronização, *P2* indica o controle do tempo, *E* a expressão de validação da sincronização e *N* a definição do momento de início da mesma. Estes elementos podem ser informados pela tabela de propriedades da figura apresentada anteriormente. No caso específico da figura teremos a seguinte construção E-LOTOS.

```
A (?x,!2) @t [t<4] start(2)
```

A tabela de propriedades permite que observemos aspectos de um mesmo elemento através de meios diferentes. Assim, se estamos interessados nas várias sincronizações de um processo, apenas observamos os gráficos que indicam sincronizações. Se estamos interessados em uma sincronização em especial usamos a tabela de propriedades para ter acesso aos dados envolvidos nesta sincronização. Em E-DART, como será apresentado posteriormente, cada elemento gráfico pode ter uma tabela de propriedades associada.

5.4 Elementos Gráficos Externos aos Retângulos de Processos

Para que a extensão de DART resulte em diagramas claros não podemos contar com a possibilidade de se introduzir novos elementos gráficos dentro dos retângulos que delimitam os processos. Isto tornaria os gráficos ilegíveis para sistemas onde os processos são complexos e possuem muitas ações de sincronização. A solução para este problema é a inclusão de novas construções externamente aos retângulos dos processos. Assim, temos o seguinte exemplo, onde antes de uma instanciação de processos existe a ocorrência de uma ação interna.

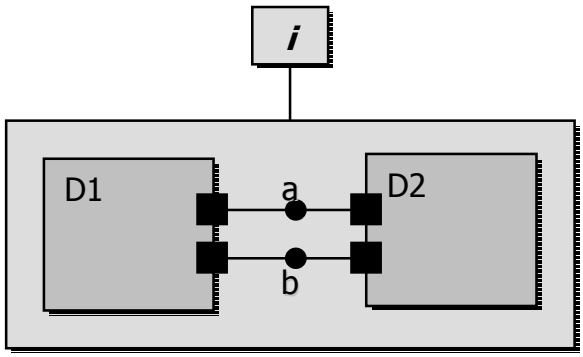


FIGURA 5.8 - Ação Interna precedendo uma instanciação

Este tipo de construção, além de não afetar a complexidade dos gráficos DART, é mais próxima de construções do tipo fluxogramas, que seguramente são apropriadas aos conhecimentos dos usuários, isto porque os fluxogramas são mais similares às construções apresentadas por linguagens de programação imperativas do que com linguagens com construções matemáticas, como é o caso de LOTOS.

Ainda que os novas construções incluídas por E-DART sejam colocadas fora dos limites do retângulo de um processo, estas construções pertencem a um processo específico. Esta informação, por sua vez, deve ser explicitamente mostrada. Temos então uma incoerência em nossas construções gráficas. Como um elemento pode pertencer a um processo se este elemento encontra-se fora dos limites do processo?

A resposta para esta questão não é simples. Primeiro porque se quisermos clareza nas especificações os novos elementos devem encontrar-se fora dos retângulos dos processos. Ao mesmo tempo estes elementos devem pertencer a um processo. Por outro lado, se analisarmos a figura anterior, o que temos na realizada é uma instanciação de dois processos *D1* e *D2* e não a definição de um processo mais externo, definido pelo retângulo que engloba *D1* e *D2*. A definição que estamos procurando refere-se a um processo que possui uma ação interna como primeiro elemento.

Desta forma, em E-DART temos a seguinte definição. Os diagramas DART são usados apenas para a instanciação de processos e não para a definição dos mesmos. A definição de processos deve ser feita por outros elementos da sintaxe gráfica. Isto permite à construção do exemplo anterior estar consistente, visto que o diagrama DART não define um processo, mas uma instanciação de processos que é precedida por uma ação interna. Assim, ao invés de dizermos “diagramas externos aos retângulos de processos” devemos dizer “diagramas externos aos retângulos de instanciação de processos”.

5.5 Elementos Gráficos E-DART

Baseados nas considerações apresentadas nas outras sessões, passamos agora a apresentar os vários elementos gráficos da sintaxe de E-DART. Novamente, cada um destes elementos pode possuir uma tabela de propriedades associada que permite o acesso direto a informações correlatas a estes elementos. Além disso, os elementos apresentados seguem uma construção que lembra os fluxogramas, tornando a sintaxe mais próxima aos conhecimentos dos usuários, que são mais “procedurais”. Por fim, os diagramas originais de DART foram restringidos e são capazes de definir apenas a instanciação de processos.

E-DART emprega sobre os elementos gráficos um esquema de cores que permite a diferenciação dos elementos não apenas pela sua forma geométrica, mas também por sua cor. O uso de cores geralmente não é usado em sintaxes gráficas, que normalmente usam apenas a diferenciação pelo fato de um elemento estar ou não preenchido. O uso de cores, além de ser mais atrativo, também torna os gráficos mais intuitivos, pois geralmente pode-se atribuir um significado a uma cor. Por exemplo, o vermelho geralmente indica uma situação onde algo é proibido. Como veremos nos diagramas apresentados, o uso de um esquema de cores só vem a melhorar a compreensão das especificações.

Os elementos apresentados nesta seção referem-se apenas a definição dos processos E-LOTOS e de seu comportamento interno. Outros elementos introduzidos por E-LOTOS, como módulos por exemplo, serão definidos posteriormente, porque de forma geral são construções que possuem elementos não representáveis da mesma forma gráfica como os elementos desta seção o são.

5.5.1 Definição de Processos

Como vimos anteriormente, a definição dos processos em E-DART será diferente da empregada em DART. Um processo em E-DART corresponde graficamente a uma área ilimitada onde encontram-se os símbolos que definem o comportamento interno destes processos.

As portas e os parâmetros de instanciação dos processos devem ser definidos em um local externa à área de definição do comportamento interno. Assim, os elementos que definem um processo encontram-se disjuntos, o que facilita a compreensão dos sistemas.

A forma de definição das portas, dos parâmetros de instanciação e da área de comportamento não são definições estritas de E-DART, sendo seu uso liberado para que sejam utilizados da forma mais conveniente. Como veremos no próximo capítulo, por exemplo, a área de comportamento de um processo no Editor E-DART é definida por uma janela de gráficos. As portas e parâmetros de instanciação, por sua vez, estão localizados dentro de uma árvore que organiza a especificação dos sistemas.

Além destes três elementos, a definição de um processo possui ainda uma tabela de propriedades para os seguintes dados: nome do processo, tipo de retorno e a informação que indica se o processo pode ser acessível por outros módulos.

Nome	ProcessX
Tipo	none
Público	false

FIGURA 5.9 - Tabela de propriedades para a definição de processos

Assim, podemos resumir a definição de processos em E-DART como sendo o conjunto dos seguintes elementos:

- Área de definição do comportamento interno
- Portas de sincronização
- Parâmetros de instanciação
- Tabela de propriedades

A definição de processos ficará mais clara a medida que outros elementos de E-DART forem sendo apresentados.

5.5.2 Sincronização

Uma ação de sincronização é definida através de um quadrado azul cujo único conteúdo é o nome da porta sobre a qual ocorrerá a sincronização. Nenhuma informação extra estará contida neste quadrado, de forma a tornar a observação do conjunto de sincronizações de um processo simples.

Entretanto, para cada sincronização em particular temos uma tabela de propriedades com os seguintes dados:

Parâmetros	""
Tempo	""
Condição	true
Início	0

FIGURA 5.10 - Tabela de propriedades para as ações de sincronização

Os *parâmetros* são os dados trocados pela sincronização. O valor padrão para os parâmetros é vazio, significando que em uma sincronização padrão nenhuma informação é trocada entre os processos envolvidos, apenas havendo a sincronização em si.

Em *tempo* temos a definição de variáveis que controlarão o tempo decorrido em uma sincronização. Normalmente nenhuma variável é definida, indicando que a sincronização não terá seu tempo controlado, a menos que se especifique tais variáveis.

Em *condição* temos uma expressão booleana que avalia a execução da sincronização. Enquanto esta expressão resultar em um valor verdadeiro (*true*) a sincronização será possível. Se no entanto tivermos uma avaliação negativa (*false*) a sincronização é interrompida e o processo que contém a sincronização evolui para uma construção *stop*, indicando o término sem sucesso. O valor *default* para a condição é *true*, o que permite que uma sincronização possa ser avaliada.

Por fim, o dado *início* indica quanto de retardo será empregado antes que a sincronização possa ser avaliada. Por *default*, este valor é zero, indicando que nenhum atraso deve ocorrer e a sincronização deve ser avaliada assim que disponível.

O exemplo a seguir corresponde a seguinte construção E-LOTOS: `s(!3,?x)@t[t<2]`

Parâmetros	!3,?x
Tempo	T
Condição	t<2
Início	0

FIGURA 5.11 - Exemplo de sincronização E-DART

Aqui, tem-se uma sincronização ocorrendo na porta *s*, onde será enviado o valor 3 e será recebido pela porta outro valor que será armazenado na variável *x*. O tempo decorrente desta sincronização será controlado pela variável *t*, sendo que a condição para a avaliação desta sincronização é a de que o tempo de duração da mesma seja menor que o valor 2, denotado pela condição *t<2*. Por fim, temos que nenhuma atraso será empregado, e a sincronização será avaliada assim que disponível.

5.5.3 Operador de atraso

Em E-DART o operador de atraso é indicado através de uma ampulheta. Tem-se também uma tabela de propriedades cujo único dado a ser especificado é o número de unidades de tempo que deve ser usado pelo operador. O valor *default* para este dado é zero, que indica a ausência de atraso na operação. A figura a seguir corresponde a construção E-LOTOS `wait(7)`.

Atraso	7

FIGURA 5.12 - Exemplo do operador de atraso em E-DART

A forma do operador de atraso é muito simples em E-DART porque este operador também é muito simples no próprio E-LOTOS, como podemos observar pelo exemplo anterior. Neste, o processo que contém o operador de atraso terá sua execução retardada por 7 unidades de tempo.

5.5.4 Ação Interna

O operador de ação interna é um dos operadores mais simples encontrados em E-DART. Sua representação é constituída apenas de um quadro com a letra ***i*** interna a ele. Nenhuma tabela de propriedades esta associada, mesmo porque este operador não possui nenhum parâmetro complementar. A figura a seguir apresenta o gráfico usado para representar as ações internas de um processo.

FIGURA 5.13 - Operador de ação interna

5.5.5 Atribuição

A atribuição foi introduzida em E-LOTOS de acordo com a seguinte construção: `P := E,` onde *P* são os parâmetros da atribuição, enquanto *E* é uma expressão de dados que será associada aos parâmetros de *P,* como vimos no capítulo de descrição de E-LOTOS. Tomemos novamente a seguinte construção como exemplo.

```
(?l, ?r) := reflect (pt)
```

Aqui, *l* e *r* receberão os valores de retorno da avaliação de *reflect,* que tem como parâmetro *pt*. Em E-DART, uma atribuição é denotada por um retângulo azul onde, internamente, tem-se a expressão de atribuição. Para este retângulo existem também uma tabela de propriedades cujos dados são os parâmetros de recebimento (à esquerda de :=) e a expressão de avaliação (à direita de :=).

No caso da atribuição tem-se uma redundância de informações, pois os dados de uma atribuição encontram-se definidos em dois lugares ao mesmo tempo: dentro do retângulo da atribuição e nos dados da tabela de propriedades. Apesar disso, veremos no próximo capítulo que esta repetição de informações nos ajudará na implementação da ferramenta.

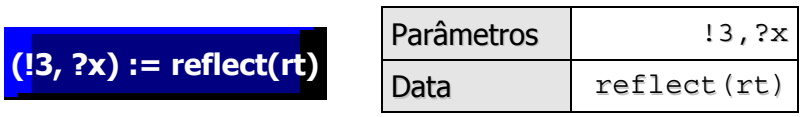


FIGURA 5.14 - Exemplo de atribuição E-DART

5.5.6 Instanciação de Processos

Como dito anteriormente, a instanciação de processos E-DART é dada de acordo com os gráficos de DART. Nesta instanciação são usados pelo menos dois retângulos, um interno ao outro. O retângulo mais externo representa o processo corrente e tem por finalidade informar as portas de sincronização deste processo. O retângulo mais interno representa o processo que se está instanciando e suas portas de sincronização. Para indicar como as portas do processo corrente se comunicará com o processo sendo instanciado são usados os pontos de sincronização.

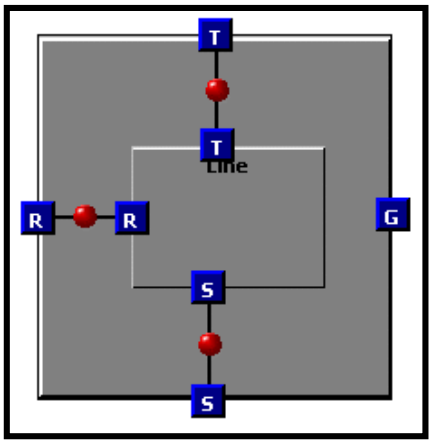


FIGURA 5.15 - Instanciação de processos em E-DART

Os pontos de sincronização que interligam as portas podem estar relacionados com portas do processo corrente e com portas de um processo instanciado, ou apenas relacionados com portas de processos instanciados. Neste caso, os pontos de sincronização interligam portas dos processos instanciados e definem indiretamente o operador “hide” de LOTOS.

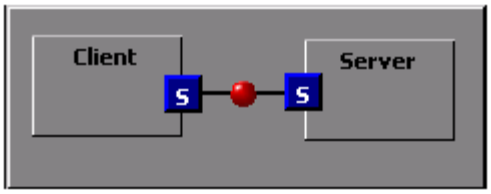


FIGURA 5.16 - Instanciação de processos com operador "hide"

Em E-DART tem-se apenas dois níveis de retângulos em uma instanciação de processos: o retângulo do processo corrente e os retângulos dos processos que serão instanciados. Em DART padrão, entretanto, em um mesmo diagrama era possível ter-se vários níveis de indentação destes gráficos. Isso era devido ao fato de que os gráficos eram usados para a definição dos processos e não sua instanciação. Em E-DART esta restrição de apenas dois níveis gráficos é coerente com a instanciação. O uso de mais níveis indicaria que os gráficos, assim como em DART, estariam sendo usados para a definição de processos, o que não é o caso.

Para um melhor entendimento do processo de instanciação definimos os seguintes termos:

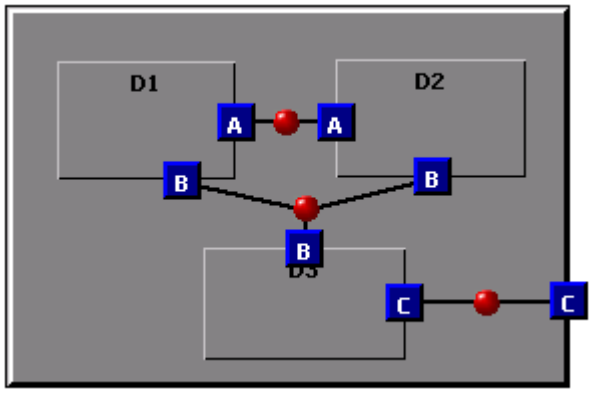
- **Retângulo de instanciação.** Refere-se ao retângulo mais externo da instanciação que representa o processo corrente e suas portas de sincronização;
- **Retângulo de processos.** Refere-se ao retângulo interno que identifica o processo que está sendo instanciado.

Em E-DART o operador de instanciação, se comparado com a sintaxe textual de E-LOTOS, é um operador de certa forma múltiplo pois corresponde a mais de um

operador E-LOTOS. Como tal, existe mais de uma tabela de propriedades para a instanciação de processos. O número de tabelas de propriedades para uma instanciação depende diretamente do número de processos que estão sendo instanciados em um retângulo de instanciação. Para cada processo instanciado tem-se uma tabela diferente. Essa diversidade no número de tabelas é necessária para que possa ser informado, a cada processo instanciado, os vários dados necessários para a execução dessa ação.

Os dados da tabela de propriedades, para cada um dos processos, também não são fixos, e dependem diretamente do número de parâmetros que o processo possui. As portas de sincronização, em uma instanciação, tem suas ligações feitas dentro do retângulo de instanciação. Entretanto, os parâmetros de cada processo devem também possuir valores na instanciação. Estes valores são informados então na tabela de propriedade, que possui uma entrada para cada parâmetro existente.

A próxima figura exemplifica uma instanciação de processos e apresenta a tabela de propriedades para cada um destes processos instanciados.



D1	
a	7
b	"send"
c	1
d	23,7

D2	
x	44
y	35

D3	
t	"send"
v	"local"
w	1004

FIGURA 5.17 - Instanciação de processos e tabelas de propriedades

5.5.7 Preempção

Como vimos na seção "Restringindo DART: Tornando a Linguagem Consistente", o operador de preempção foi retirado de DART para tornar a linguagem consistente com nossa proposta. Precisamos agora definir um novo operador de preempção que seja coerente com os outros operadores.

Assim, E-DART define o operador de preempção como sendo um par de retângulos com símbolo → interno a estes retângulos. É empregado no operador um par de retângulos de forma a delimitar o escopo do operador. Um primeiro retângulo

define o início das expressões envolvidas na preempção, enquanto que o segundo retângulo define o final destas expressões.

As expressões envolvidas na preempção são então, de certa forma, envolvidas pelo operador. Existem dois "ramos" de expressões. Um ramo corresponde a expressão avaliada normalmente, e o outro ramo corresponde a expressão que pode interromper a primeira.

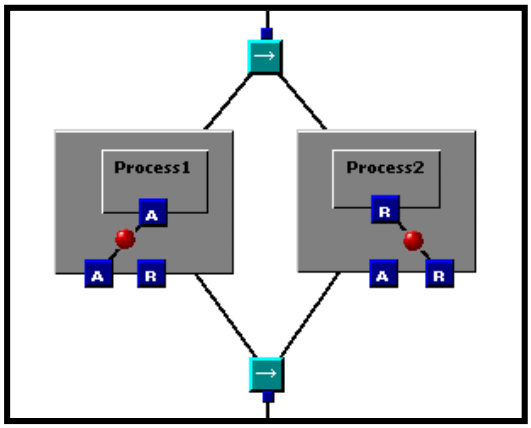


FIGURA 5.18 - Operador de preempção

O ramo da esquerda corresponde a expressão avaliada normalmente enquanto o ramo da direita à expressão que pode interromper a primeira.

Por fim, devemos notar que não existe nenhuma tabela de propriedades associada ao operador de preempção, visto que toda a informação existente é obtida diretamente dos diagramas.

5.5.8 Escolha

O operador de escolha é bastante semelhando ao operador de preempção, no que diz respeito a sua estrutura gráfica. O operador é definido também por dois retângulos que delimitam o escopo do operador. Dentro destes retângulos encontra-se o símbolo de interrogação (?). Assim como na preempção, não existe tabela de propriedades associada ao operador de escolha.

Entre estes retângulos encontram-se as expressões que podem ser avaliadas em uma escolha. No operador de preempção existem dois ramos de expressões. No operador de escolha existem tantos ramos quanto forem o número de expressões possíveis de serem avaliadas. A figura a seguir exemplifica a utilização do operador de escolha.

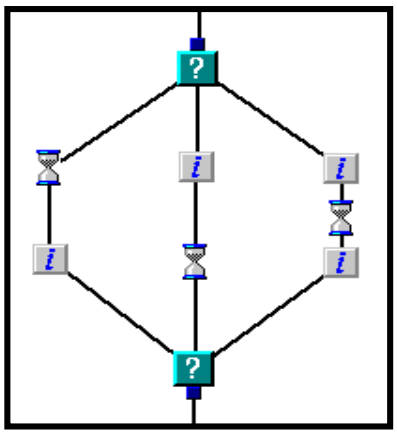


FIGURA 5.19 - Operador de escolha

Aqui temos três expressões que podem ser avaliados dentro do operador de escolha. A ordem em que estas expressões estão dispostas não influi na semântica da escolha, como acontecia com o operador de preempção.

5.6 Considerações sobre E-DART

As construções apresentadas neste capítulo são referentes principalmente a definição dinâmica do comportamento dos processos. E-LOTOS introduz também melhoramentos em relação a definição tipos abstratos de dados. Em relação a este aspecto nenhuma notação gráfica foi criada em E-DART por não ser a definição de dados o objetivo principal da sintaxe gráfica. Estamos preocupados em melhorar a expressão de comportamentos e não dos dados envolvidos nestes comportamentos.

Estendendo estas considerações, outro aspecto que deve ser considerado, além de definição dos tipos abstratos de dados, é a definição das funções. De acordo com a norma de E-LOTOS, funções são similares à definição de processos. Entretanto, as funções não possuem portas de sincronizações e não podem exprimir comportamentos que modifiquem o ambiente dos sistemas, sendo capazes apenas de executar transformações baseadas nos parâmetros de entrada. Assim, em E-DART a definição de funções não possui correspondentes gráficos, novamente, porque estamos interessados na expressão de comportamento. Como as funções não definem comportamentos, as mesmas continuam sendo definidas de forma textual.

E-LOTOS introduz também construções para o suporte a modularização das especificações. A modularização em si não define comportamentos, mas ajuda na organização das especificações além de permitir o reuso de partes de especificações. Os módulos podem constituir bibliotecas. Novas especificações podem ser construídas baseadas nos módulos armazenados em uma biblioteca. Apesar de E-DART não possuir construções gráficas referentes a modularização, no próximo capítulo poderemos observar como a modularização pode ser bem aproveitada através de estruturas em um ambiente de especificação.

Por fim, cabe dizer que E-DART, assim como DART, não é completo. Isto é, não suporta todas as construções E-LOTOS existentes. Entretanto, a linguagem, se comparada a DART e G-LOTOS, possui o maior grau de facilidade de expansão, o

que garante a E-DART o poder de ser expandido ainda mais até possuir todas as construções E-LOTOS.

6 Editor E-DART: A Ferramenta de Especificação para E-DART

O uso da sintaxe E-DART proposta neste trabalho, depende da existência de uma ferramenta que suporte esta sintaxe. Apenas a definição da linguagem, feita no capítulo anterior, não garante a sua eficiente utilização. Os melhoramentos alcançados em E-DART podem ser ainda mais úteis se pudermos contar com todas as funcionalidades que um ambiente de especificação pode disponibilizar. De nada adianta termos E-DART se continuarmos a especificar os sistemas em folhas de papel, apenas utilizando a sintaxe. O processo continuará sendo insatisfatório.

Assim, este capítulo apresenta o Editor E-DART, um ambiente gráfico orientado a janelas onde é possível a especificação de sistemas através de E-DART. O ambiente utiliza a sintaxe gráfica na construção dos processos, mas várias outras funcionalidades são usadas para melhorar a interação com usuário e tronar o processo de especificação mais intuitivo e rápido.

Na figura 6.1 é apresenta a interface gráfica da ferramenta, indicando as principais funcionalidades presentes.

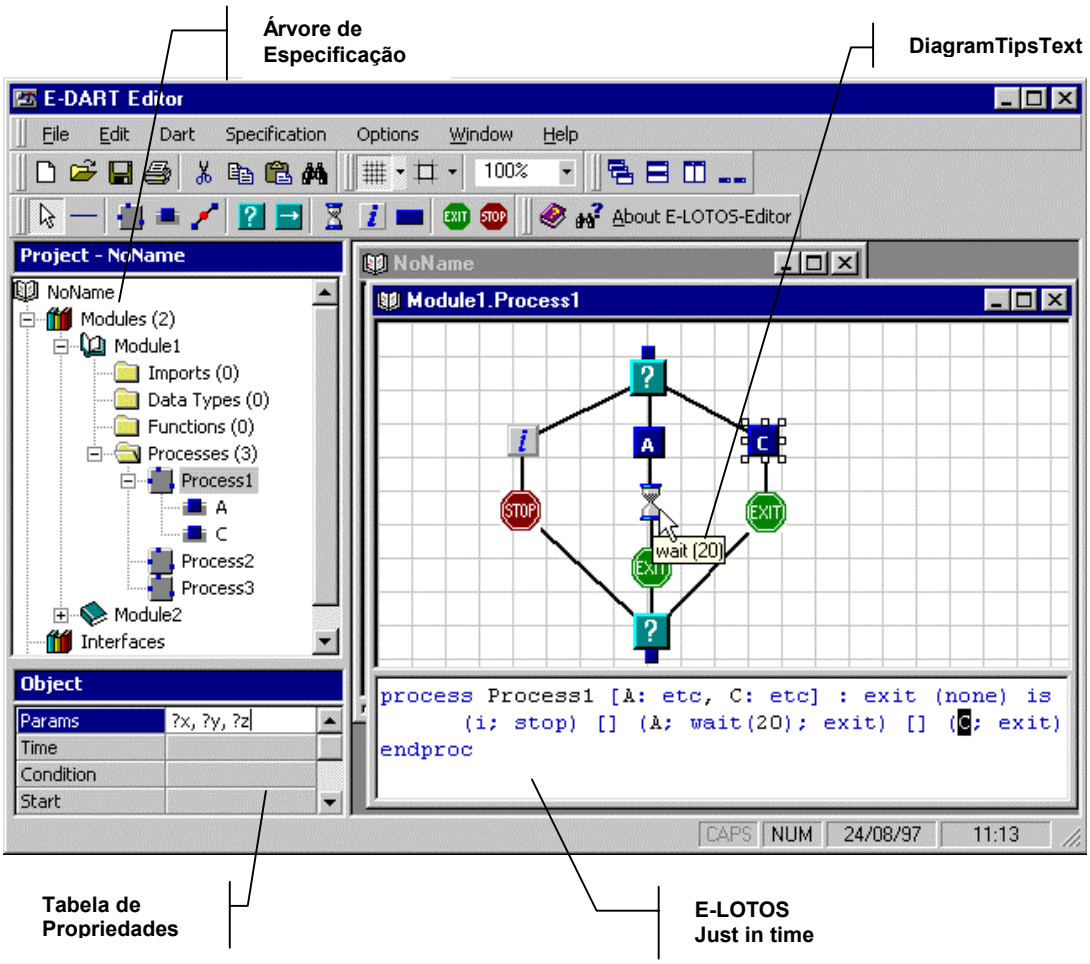


FIGURA 6.1 - Interface gráfica do Editor E-DART

As seções a seguir mostram o funcionamento do Editor E-DART e apresentam as funcionalidades existentes, indicando o modo como as mesmas são utilizadas na criação de especificações E-DART.

6.1 Organizando as Especificações

Com a introdução em E-LOTOS do conceito de modularização, uma especificação é dada por um conjunto de módulos que definem seus processos. O comportamento de um sistema é então o conjunto dos comportamentos de cada processo que constitui o sistema. Assim, uma especificação é organizada em módulos, que além de tornar a estrutura da especificação mais clara constituem-se no mecanismo utilizado para o reuso de código.

Seguindo esta perspectiva, o ambiente organiza as especificações E-DART também através de uma estrutura de módulos. Estes módulos, por sua vez, são mantidos e organizados no editor pela chamada árvore de especificação.

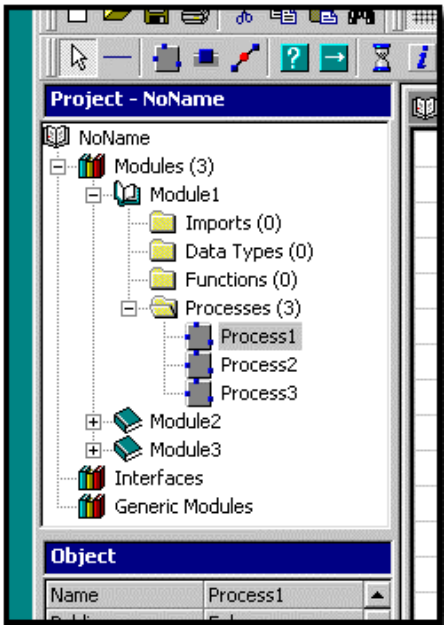


FIGURA 6.2 - Árvore de especificação

Como podemos ver na figura anterior, a árvore de especificação possui uma raiz que corresponde a especificação como um todo. Desta raiz temos três sub-árvores principais que definem os módulos, interfaces e os módulos genéricos. Atualmente a ferramenta suporta apenas a definição dos módulos, enquanto as interfaces e módulos genéricos serão suportados em versões futuras.

Dentro da sub-árvore de módulos temos outras sub-árvores, uma para cada módulo da especificação. Na figura anterior podemos notar a existência de três módulos: *Module1, Module2* e *Module3*. Cada módulo define um conjunto de tipos abstratos de dados, funções e processos. Além disso, tem-se a definição de referências

a outros módulos. Isto é feito através do nodo *Imports*. Neste nodo são definidos quais outros módulos serão referenciados. A referência pode ser em relação às funções, tipos abstratos de dados ou processos.

Logo, podemos resumir a especificação de um módulo pela ferramenta como sendo a definição dos seguintes conjuntos:

- *Imports*: módulos externos referenciados;
- *Data types*: tipos de dados definidos pelo módulo;
- *Functions*: funções definidas pelo módulo; e
- *Processes*: processos definidos pelo módulo.

Estes elementos, como pode ser visto na última figura, constituem-se em pastas na sub-árvore dos módulos. Cada uma destas sub-árvores possuem itens que definem sua constituição. Assim, a sub-árvore *Imports* possuirá um nodo para cada módulo externo referenciado; a sub-árvore *functions* possuirá um nodo para cada função definida pelo módulo.

Destas sub-árvores a mais importante é aquela que define o conjunto de processos do módulos, isto é, a sub-árvore *processes*. Na figura anterior temos três processos para *Module1*: *Process1, Process2* e *Process3*. Cada nodo que define um processo, diferentemente dos nodos de funções, tipos de dados e referências externas, são também sub-árvores.

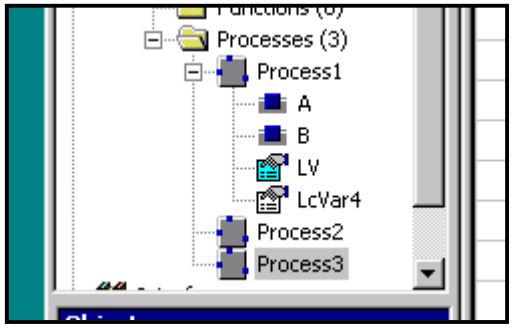


FIGURA 6.3 - Sub-árvore de processos

Uma sub-árvore de processo define três elementos distintos associados a cada processo:

- As portas de sincronização que cada processo possui para comunicar-se com o seu ambiente;
- Os parâmetros de instanciação dos processos; e
- As variáveis locais, cujo escopo é delimitado pelo processo

Com todos estes níveis de relação entre os elementos de uma especificação, a árvore de especificação permite uma organização mais clara dos sistemas, além de ser

coerente com o modelo de modularização apresentado por E-LOTOS. Do nodo raiz até os elementos de um processo, o acesso aos diversos elementos de uma especificação é rápido pois o usuário precisa apenas navegar pela árvore e acessos os itens desejados.

6.2 Definindo comportamentos

A árvore de especificação serve para organizar e tornar o acesso aos elementos de uma especificação mais rápido. Além disso, é através da árvore de especificação que é especificada a interface de um processo, isto é, as portas e os parâmetros de instanciação. Entretanto, apenas a interface de um processo não é suficiente para a definição completa de um processo. Necessitamos ainda da definição do comportamento dinâmico do mesmo. Para isso tem-se as janelas de comportamento e a caixa de ferramentas E-DART.

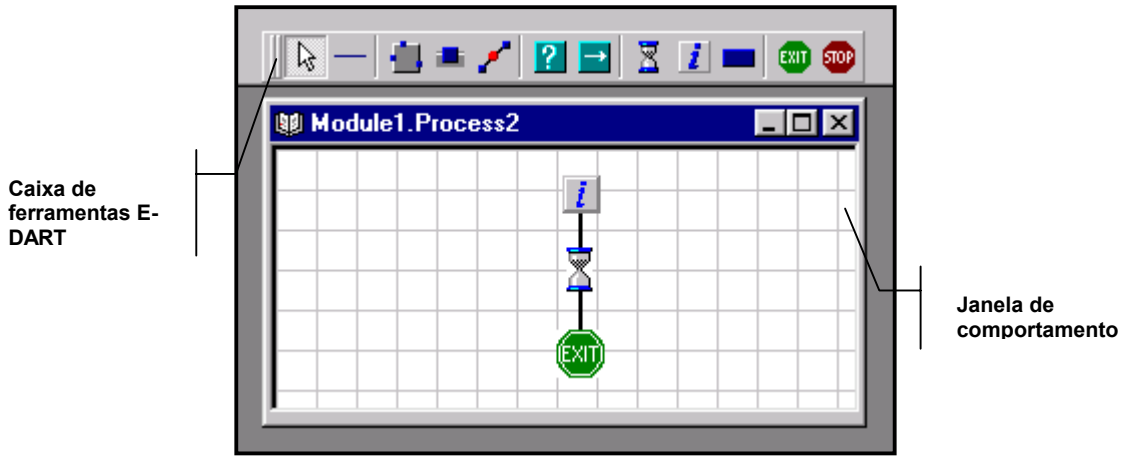


FIGURA 6.4 - Caixa de ferramentas E-DART e Janela de comportamento

Na janela de comportamento serão inseridos os símbolos gráficos E-DART que definirão o comportamento de cada processo. É a janela de comportamento que efetivamente implementa a sintaxe de E-DART, pois suporta todas as construções apresentadas no capítulo anterior. A caixa de ferramentas E-DART, localizada na parte superior do ambiente, possui todas as construções que podem ser usadas dentro de uma janela de comportamento. O usuário seleciona uma estrutura na caixa de ferramentas e logo em seguida inclui esta estrutura em uma janela de comportamento.

Para cada sistema especificado, tem-se apenas uma caixa de ferramentas, mas várias janelas de especificação. Existe uma janela para cada processo do sistema. As janelas podem estar visíveis todas ao mesmo ou pode-se escolher quais visualizar. O acesso a estas janelas se dá através da árvore de especificação. Clicando-se duas vezes no nodo de um processo temos o surgimento da janela correspondente àquele módulo clicado.

Como o Editor E-DART é um ambiente que segue a filosofia de aplicações MDI (*Multi Document Interface*) as janelas de comportamento podem ser abertas e fechadas de acordo com a necessidade do usuário. Pode-se organizar estas janelas

verticalmente ou horizontalmente como acontece em ferramentas padrão Microsoft Windows.

6.3 Propriedades das Construções E-DART

Na definição de E-DART, quase todos os símbolos gráficos possuíam uma tabela de propriedades que permitia acessar valores associados a estes símbolos. O mesmo acontece no Editor E-DART. Abaixo da árvore de especificação encontra-se a tabela de propriedades do ambiente.

Sempre que um elemento gráfico possuir atributos estes estarão acessíveis através da tabela de propriedades. Assim, clicando-se em um gráfico E-DART de uma janela de comportamento, digamos um símbolo de atraso (denotado pela ampulheta), surgirá na tabela de propriedades o atributo que indica quantas unidades de tempo serão usadas no atraso.

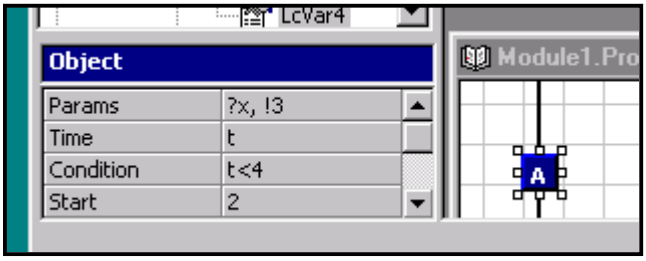


FIGURA 6.5 - Tabela de propriedades para um elementos E-DART

Apesar da tabela de propriedades ter uso imediato na implementação do acesso aos atributos de cada diagrama E-DART, as funcionalidades implementadas na tabela permitem acesso também aos parâmetros de qualquer elemento do ambiente. Isto é, a tabela de propriedades não é restrita aos gráfico E-DART, sendo possível acessar propriedades de outros elementos. Por exemplo, através da tabela de propriedade podemos modificar os tipos de dados que são suportados por uma porta de processos. Para isto, selecionamos esta porta na árvore de especificação e alteramos este atributo na tabela de propriedades.

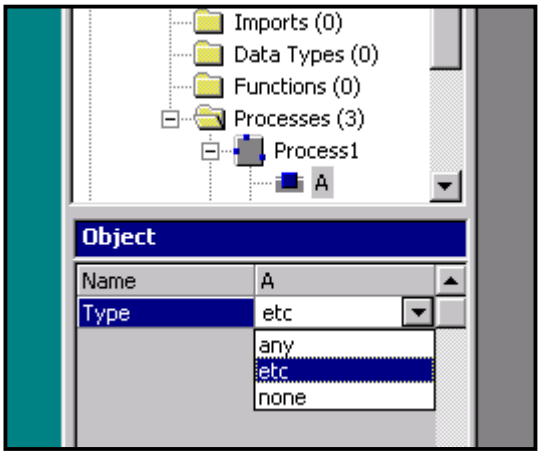


FIGURA 6.6 - Tabela de propriedades para um elemento do ambiente

Até este momento temos definido os três elementos principais que permitem a definição dos sistemas através do Editor E-DART:

- Árvore de especificação;
- Janelas de comportamento; e
- Tabela de propriedades

Estes elementos permitem o efetivo suporte à sintaxe E-DART pelo ambiente. Com isto, já é possível a especificação dos sistemas. Se nenhuma outra funcionalidade existisse, ainda assim E-DART seria possível no ambiente. Entretanto, o Editor E-DART possui funcionalidades complementares que permitem um melhor processo de especificação. A partir de agora passamos a explicar estas outras funcionalidades.

6.4 Armazenando, Recuperando e Reutilizando Módulos

As especificações criadas podem ser obviamente armazenadas em arquivos para sua posterior recuperação. Existe, entretanto, duas formas distintas para o armazenamento das especificações. Estas formas distintas estão relacionadas com os módulos dos sistemas.

A primeira forma de armazenamento é trivial. Uma especificação é salva em um único arquivo que conterá todos os módulos do sistema bem como seus processos e outros elementos. A recuperação da especificação para esta forma de armazenamento também é trivial. Lendo-se o arquivo a especificação é novamente construída pela ferramenta permitindo ao usuário alterar os dados gravados.

A segunda forma de armazenamento, por outro lado, leva em conta o fator modularização. Nesta forma os módulos de um sistema podem ser armazenados de forma independente em vários arquivos separados. Para cada módulo de uma especificação temos um arquivo correspondente. A especificação como um todo também possuirá um arquivo correspondente. Entretanto, este arquivo possuirá pouca informação, mas basicamente o conteúdo mais importante são os dados relativos aos módulos que constituem a especificação. Tem-se apenas referências a estes módulos, mais especificamente aos arquivos que armazenam os módulos.

A recuperação de uma especificação armazenada com módulos independentes é um pouco mais complexa. Primeiro é lido o arquivo principal que organiza a especificação e referencia os módulos constituintes. Para cada um destes módulos tem-se então a leitura do arquivo correspondente. Ao final, as referências internas da especificação são refeitas e a especificação reconstituída.

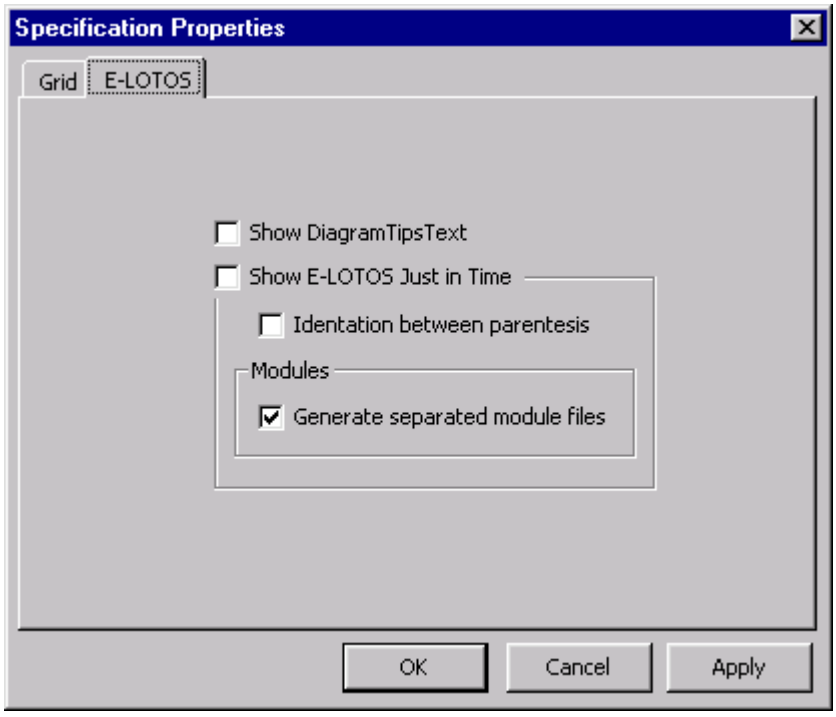


FIGURA 6.7 - Customização da forma de armazenamento

Obviamente a fato de termos duas formas distintas de armazenamento tem sua razão de existir. Com o armazenamento orientado à módulos a ferramenta implementa a funcionalidade de reuso de módulos. Isto porque é possível importar um módulo específico para dentro de uma especificação. Como os módulos podem ser armazenados independentemente, eles podem também ser importados independentemente. Assim, uma nova especificação pode ser construída através do reuso de vários outros módulos definidos anteriormente por outras especificações. Além da vantagem óbvia do menor tempo de desenvolvimento, o reuso de módulos traz mais segurança às novas especificações. Um módulo construído e testado de forma correta por uma especificação não precisa ser novamente testado pelas especificações que venham a usar este módulo.

6.5 Funcionalidades Imediatas

O acesso as funcionalidades é feito principalmente pelo uso dos menus e caixas de ferramentas localizados na parte superior do ambiente. Estes menus e caixas de ferramentas podem, de acordo com a preferência do usuário, serem movidos para outras localidades do ambiente. Podem se localizar na parte inferior ou nos lados da janelas principal, ou ainda tornarem-se caixas flutuantes.

Através dos menus a caixas de ferramentas tem-se acesso a todas as funcionalidades existentes. Entretanto, estas funcionalidades podem ser ou não aplicadas aos elementos desejados. Para uma interatividade mais rápida do usuário com os elementos da especificação foram introduzidos no ambiente os menus sensíveis ao contexto. Tais menus são acessíveis através do botão direito do mouse. Clicando-se com este botão sobre um determinado elementos um menu sensível ao contexto é apresentado e mostra todas as operações que podem ser executadas sobre o elemento clicado.

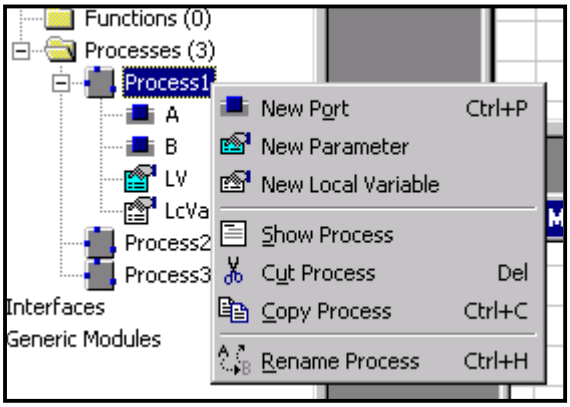


FIGURA 6.8 - Menus sensíveis ao contexto

No exemplo acima vemos o menu existente para um processo na árvore de especificação. O uso de menus sensíveis ao contexto agiliza a execução de funções que normalmente são disparadas após uma navegação mais extensa sobre as opções dos menus convencionais. Além disso, os menus sensíveis ao contexto auxiliam no processo de aprendizado de novos usuários. Como os itens destes menus são as opções possíveis sobre um determinado elemento, o novo usuário assimila de forma mais fácil o que pode ser feito ao não com este elemento.

6.6 Tradutor E-LOTOS: Obtendo a Sintaxe Textual das Especificações

O uso de E-DART não inviabiliza de forma alguma o uso de E-LOTOS. O principal objetivo de E-DART é livrar o usuário das complexidades de E-LOTOS. Entretanto, usuários mais avançados e familiarizados com técnicas de descrição formal podem achar conveniente o acesso a código E-LOTOS correspondente aos diagramas E-DART.

O Editor E-DART possui um tradutor E-DART/E-LOTOS capaz de gerar tais textos E-LOTOS para as especificações criadas. Este tradutor é percebido pelo usuário através de três funcionalidades principais apresentadas a seguir.

6.6.1 DiagramTipsText

Em ferramentas modernas, quando o usuário repousa o mouse sobre um botão de interface por um determinado período de tempo, aparece próximo ao mouse uma pequena caixa de texto que descreve de forma muito resumida a funcionalidade daquele botão. Esta funcionalidade é chamada *ToolTipsText*, que poderia ser traduzida como “texto de dicas de ferramenta”.

O Editor E-DART suporta esta funcionalidade de forma padrão para todos os elementos de interface que possuam alguma operação associada, caso principal dos botões das caixas de ferramentas. Além disso, o ambiente estende esta funcionalidade para os diagramas E-DART na janela de comportamento dos processos. Se o usuário repousar o mouse sobre um elemento E-DART, após alguns instantes surgirá próximo ao mouse uma janela de texto que conterá uma porção de código E-LOTOS correspondente àquele elemento.

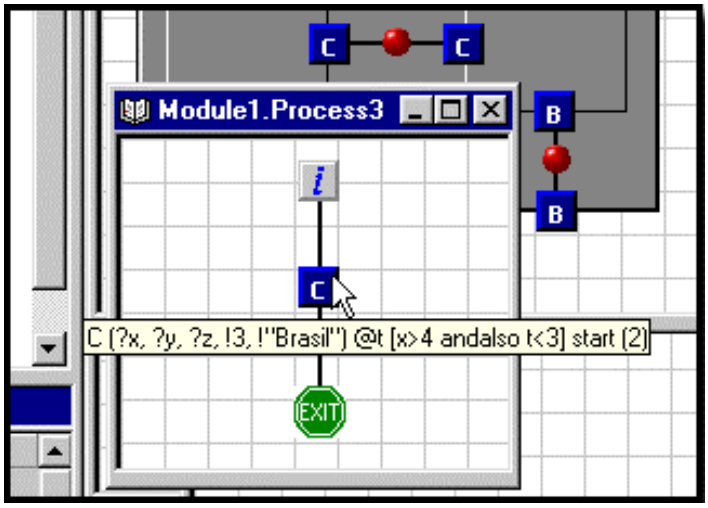


FIGURA 6.9 - DiagramTipsText

A esta nova funcionalidade foi dado o nome de *DiagramTipsText*. Na figura acima podemos observar a ativação da funcionalidade sobre uma operação E-DART de sincronização sobre a porta *C* do processo *Process3*. O texto apresentado informa toda a construções E-LOTOS correspondente ao gráfico E-DART.

As *DiagramTipsText* são úteis quando estamos interessados em uma ação particular de um processo. Outro fato importante é a facilidade existente na obtenção do código E-LOTOS para este elemento por parte do usuário. Não existe navegação em menus convencionais ou sensíveis ao contexto. Não é necessário nem mesmo a seleção do elemento em que estamos interessados. Basta apenas apontá-lo e esperar por alguns instantes.

6.6.2 E-LOTOS Just in Time

As *DiagramTipsText* apresentam código E-LOTOS para apenas um diagrama E-LOTOS específico. Entretanto, muitas vezes os usuários podem estar interessados em ter acesso ao código E-LOTOS de um processo inteiro. Isto é possível através da funcionalidade *E-LOTOS Just in Time*.

Na parte inferior da janela de comportamento dos processos existe uma região que apresenta apenas texto, oculta por default. Se o usuário desejar, esta área pode tornar-se visível e passa a apresentar o código E-LOTOS do processo correspondente. É interessante notar que a atualização deste código E-LOTOS é instantânea. Qualquer modificação na especificação que possa representar uma alteração em código E-LOTOS é imediatamente sentida na área textual. Isto garante que o código apresentado encontra-se sempre consistente em relação a especificação como um todo.

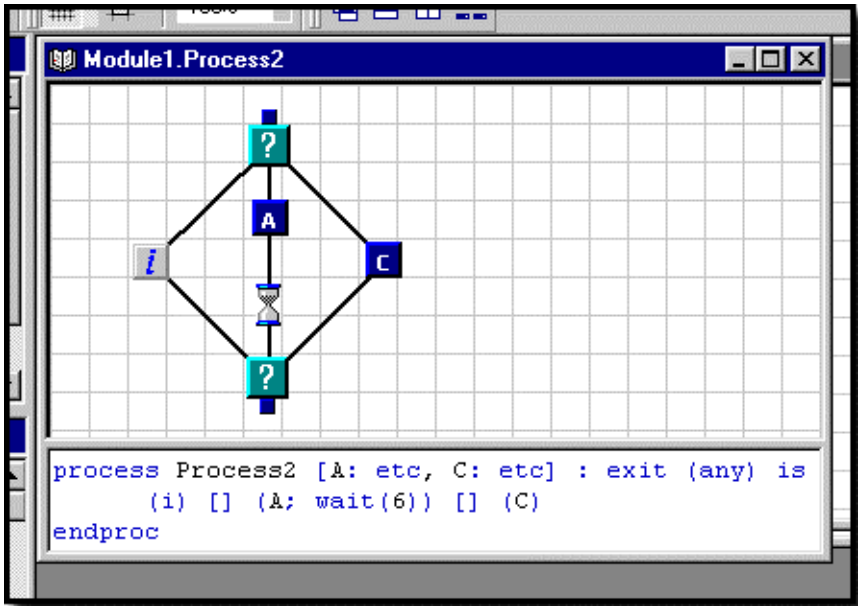


FIGURA 6.10 - E-LOTOS Just in Time

Manter o código E-LOTOS constantemente atualizado é possível porque o tradutor E-DART/E-LOTOS do ambiente é executado toda vez que uma alteração é realizada. O tradutor, quando requisitado atualiza todos os pontos modificados, o que garante a consistência permanente.

6.6.3 Armazenamento E-LOTOS

As *DiagramTipsText* geram E-LOTOS no nível de elementos específicos. *E-LOTOS Just in Time*, respondem por código E-LOTOS a nível de processos. Por fim, se quisermos código E-LOTOS para toda uma especificação devemos utilizar esta última funcionalidade: o armazenamento E-LOTOS.

Quando executamos a função de salvamento de uma especificação os arquivos gerados obedecem um padrão interno do ambiente de armazenamento dos dados. Este padrão garante a posterior recuperação das especificações criadas. Entretanto, podemos escolher, na momento do salvamento, a forma de armazenamento desejado: ou especificações E-DART (*.spc) ou código E-LOTOS (*.els).

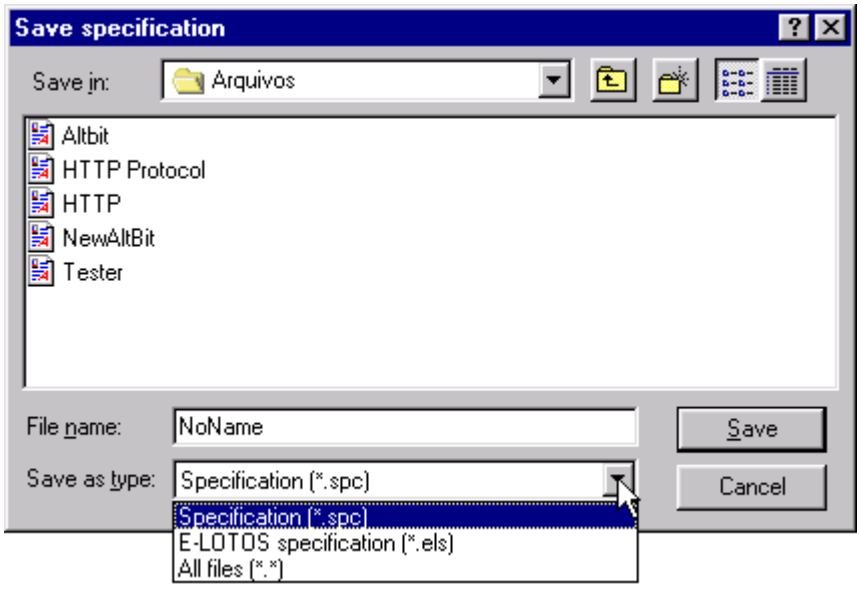


FIGURA 6.11 - Armazenamento E-LOTOS

O armazenamento E-LOTOS gera arquivos texto que contém toda a especificação E-DART em texto E-LOTOS. Estes arquivos podem então ser utilizados por outras ferramentas para validação, teste ou animação da especificação. O único inconveniente no armazenamento E-LOTOS é o fato de que o Editor E-DART ainda não é capaz de construir gráficos E-DART a partir de fontes textuais. Ainda que isto fosse possível, a reconstituição E-DART não seria perfeita, pois em um arquivo E-LOTOS não são armazenadas informações gráficas como posicionamento e dimensões dos elementos E-DART.

6.7 A Implementação do Editor

O Editor E-DART foi implementado utilizando conceitos e técnicas desenvolvidos para este tipo de ferramentas [GRA 95, GRA 97A, GRA 97B]. As maiores influências foram sobre o tratamento e manipulação dos gráficos E-DART.

O Editor E-DART foi escrito no ambiente Microsoft Visual Basic 5.0. Outras opções consideradas para o desenvolvimento da ferramenta foram o Borland C++ 5.0, Borland C++ Builder, Borland Delphi e Microsoft Visual C++ 5.0. Os fatores que mais influíram na escolha do ambiente foram: tempo de desenvolvimento, facilidade de alteração e manutenção de sistema, confiabilidade da linguagem de programação e facilidade no uso de componentes de software de terceiros.

Os ambientes C++ considerados têm seu ponto forte no emprego da linguagem C++, que permite um maior controle sobre as APIs de programação do Windows, além de ser um linguagem verdadeiramente orientada a objetos. Entretanto, os ambientes C++ possui um processo de compilação e linkagem demasiadamente lento, o que prejudica muita o processo de desenvolvimento. Pequenas alterações nos códigos fonte corresponde a vários minutos de espera na compilação. Este foi o principal motivo para que estes ambientes fossem abandonados.

Outro motivo também importante foram certas características conhecidas da linguagem C que tornam o processo de desenvolvimento mais lento. Como a linguagem é extremamente poderosa, poucas consistências são feitas sobre o código fonte, o que muitas vezes leva a erros indesajados. Um exemplo clássico disso é a manipulação de strings. O usuário é responsável por gerenciar a memória que o conteúdo de uma string pode ocupar. Se tal memória for excedida um erro ocorre. A simples preocupação com este procedimento já torna o processo de desenvolvimento mais lento.

Ainda cabe ressaltar a dificuldade no uso de componentes de software em ambiente C++. Excetuando-se o Borland C++ Builder, os outros ambientes não fornecem uma grande facilidade de uso de componentes ActiveX nos projetos.

A escolha final ficou então entre o Visual Basic e o Delphi. A linguagem Object Pascal suportada por Delphi é notadamente superior ao Visual Basic. Ela possui conceitos de orientação a objetos bem mais sofisticados, o que permite uma modelagem mais consistente dos programas. Em relação à facilidade no uso de componentes tanto Visual Basic quanto Delphi apresentam um suporte muito bom. Delphi tem foco principal em componentes construídos com Object Pascal, enquanto Visual Basic em componentes ActiveX. Por fim, em relação ao ambiente de programação em si, Visual Basic apresenta-se ligeiramente melhor, implementado funcionalidades muito úteis que aceleram o tempo de desenvolvimento.

O que decidiu a escolha por Visual Basic foram basicamente dois fatores. O primeiro diz respeito ao uso de controles ActiveX. Visual Basic fornece um melhor suporte a este padrão de Delphi. Como existem uma infinidade de controles ActiveX disponíveis no mercado, o tempo de desenvolvimento do sistema é reduzido, porque na maior parte das vezes encontramos controles para quase todas as nossas necessidades. Assim, preferimos ActiveX e por conseqüência Visual Basic.

O segundo fator diz respeito a facilidade na criação dos próprios controles ActiveX necessários ao ambiente, mas que não eram disponibilizados por terceiros. Neste caso tais controles deveriam ser desenvolvidos. Neste sentido o suporte ao desenvolvimento de controles ActiveX do Visual Basic é notadamente superior a Delphi.

Apesar de apresentarem uma abordagem orientada a objetos um pouco incompleta, pois não permitem por exemplo herança, a utilização de componentes ActiveX permite um desenvolvimento rápido e seguro. Além disso, a grande gama de componentes disponíveis na Internet permite um visual mais profissional da ferramenta. Um exemplo disso são os menus utilizados que seguem o estilo das ferramentas do pacote Microsoft Office 97.

O desenvolvimento do ambiente deu-se em duas principais etapas: o tratamento de gráficos e o gerenciamento das especificações. Uma descrição mais

técnica e detalhada sobre o desenvolvimento da ferramenta é fornecida no anexo B: "Documentação sobre a Implementação do Editor E-DART"

6.7.1 O Tratamento de Gráficos

Nesta etapa a preocupação maior era a criação de um componente ActiveX que implementasse uma área gráfica onde seria possível inserir-se figuras quaisquer. Estas figuras poderiam ser deslocadas e terem sua dimensões alteradas. Várias figuras poderiam ser selecionadas ao mesmo tempo e movidas em conjunto.

Além disso, deveria existir uma hierarquia entre os elementos de forma a permitir que uma figura pudesse "pertencer" a outra. Neste caso, a movimentação de uma figura implicaria também na movimentação da outra. Por fim, as figuras poderiam ser conectadas por linhas identificando algum tipo de relação entre figuras.

Estas considerações sobre como o componente deveria tratar os gráficos são baseadas na forma como os diversos elementos gráficos E-DART se relacionam um com os outros. Um exemplo disso pode ser observado na instanciação de processos, que possui duas figuras de tipos distintos: o processo instanciado e o elemento de instanciação. Quando o processo instanciado fosse movido nada deveria acontecer com o elemento de instanciação. Entretanto, o movimento deste último deveria gerar também o movimento do processo instanciado.

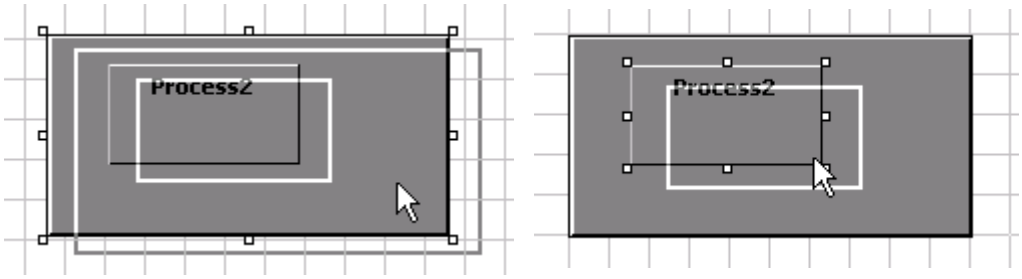


FIGURA 6.12 - Movimentação de figuras em uma instanciação de processos

O componente criado foi então incluído em todas as janelas de comportamento para suportar a definição dos comportamentos dos processos das especificações.

6.7.2 Gerenciamento das Especificações

De posse de todos os componentes necessários para a implementação do ambiente o próximo passo foi a elaboração de como uma especificação seria representada internamente no ambiente. Assim, cada elemento E-DART tornou-se uma classe de objetos Visual Basic. Da mesma forma aconteceu com elementos de gerenciamento da especificação, como por exemplo os nodos da árvore de especificação.

A seguir apresentamos as classes implementadas para suportar o gerenciamento das especificações.

- Action. Modela o operador de atribuição.
- Connection. Modela um ponto de sincronização.
- DataType. Modela um tipo abstrato de dados textual.
- Decision. Modela o operador de decisão.
- Disable. Modela o operador de preempção.
- ElotosFunction. Modela uma função em E-LOTOS textual.
- ExitAcition. Modela a terminação com sucesso de processos.
- IAction. Modela o operador de ação interna não observável.
- LocalVariable. Modela instâncias de variáveis locais aos processos.
- Module. Modela um nodo na árvore de gerenciamento para representar um módulo.
- Parameter. Modela os parâmetros de instanciação dos processos.
- Port. Modela uma porta de sincronização.
- PortInstance. Modela uma referência a porta na instanciação de processos.
- Process. Modela um processo.
- ProcessInstance. Modela uma referência a um processo na instanciação de processos.
- Specification. Modela uma especificação.
- StopAction. Modela a terminação sem sucesso de processos.
- Sync. Modela uma sincronização em uma porta.
- Wait. Modela o operador de atraso.

7 Conclusão e Trabalhos Futuros

A presente dissertação de mestrado apresentou uma proposta gráfica para a técnica de descrição formal E-LOTOS. A proposta gráfica é baseado em G-LOTOS e DART, outras duas abordagens que forma criadas para a definição de sistemas LOTOS. A escolha de E-LOTOS se deve a necessidade do suporte à noção de tempo quantitativo nas especificações de sistemas que tem no tempo um fator principal.

E-DART, a nova sintaxe gráfica proposta, vêm solucionar inicialmente dois principais problemas em relação a LOTOS: a inexistência de sintaxe gráfica para uma extensão temporal (E-LOTOS) e a grande complexidade das construções textuais da linguagem, extremamente matemática. Através de E-DART pode-se definir ações que levem a noção de tempo em consideração, sempre de acordo com a sintaxe textual. Mas a principal vantagem de E-DART é que o usuário pode abstrair, no processo de especificação, as complexidades inerentes à sintaxe textual citadas. E-DART torna o uso de E-LOTOS mais simples, além de adaptar a linguagem à uma forma de visualização das especificações de sistemas mais próxima dos conceitos aos quais os usuários estão acostumados. Enquanto E-LOTOS continua tendo uma estrutura extremamente matemática, E-DART apresenta gráficos de uma forma seqüencial, assemelhando-se a fluxogramas.

O uso de E-DART entretanto deve ser estimulado, pois a sintaxe por si só não garante seu efetivo emprego. É necessário fornecer meios eficientes para que os usuários se sintam confortáveis no emprego da proposta gráfica. Para isso foi criado o Editor E-DART, uma ambiente gráfico orientado a janelas que permite o extensivo uso de E-DART através de funcionalidades de interação com o usuário avançadas.

O grande mérito do ambiente é auxiliar o usuário na especificação dos sistemas. Para isso, o Editor E-DART possui um modelo de gerenciamento de especificações hierárquico, que inclusive implementa os conceitos de modularização introduzidos por E-LOTOS. A forma como o armazenamento das especificações são armazenadas permitem um eficiente reuso de módulos por novas especificações, o que reduz o tempo de desenvolvimento e teste de novos sistemas.

Para usuários familiarizados com E-LOTOS o ambiente prove mecanismo interativos de acesso a código textual E-LOTOS para as especificações. Este acesso se dá em três níveis distintos: a nível de ação, processo e especificação. Um tradutor E-DART/E-LOTOS interno ao ambiente garante a constante consistência do código, permitindo a atualização automática para cada modificação executada. Assim, o Editor E-DART é um capacitor das vantagens introduzidas pela sintaxe gráfica E-DART.

Assim, as implementações apresentadas hoje em dia mostram-se coerentes com a idéia de simplificar E-LOTOS, permitindo um uso mais extensivo da proposta. Os gráficos ajudam muito neste aspecto e o Editor E-DART efetivamente disponibiliza meios para que E-LOTOS seja utilizado.

Por este ponto de vista, o ambiente apresentado cumpre sua missão tornando E-DART viável. Por outro lado, se considerarmos operações como validação de especificações, animação do comportamento de processos e interação com outras ferramentas, percebemos que o leque de funcionalidades possíveis abre-se mais ainda. Neste aspecto, novas versões do Editor E-DART são previstas para a implementação

de funcionalidades que dêem suporte a estas novas operações sobre as especificações de sistemas.

Tradutor E-LOTOS/E-DART

Como descrito neste trabalho, o Editor E-DART é capaz de gerar código textual E-LOTOS através do armazenamento de especificações em arquivos texto. Entretanto, a recuperação de tais especificações não é possível pois não existe um tradutor E-LOTOS/E-DART na ferramenta.

Numa primeira instância um tradutor se faz necessário para sanar esta deficiência do ambiente, mas o maior objetivo é permitir que outras especificações E-LOTOS, geradas por outras ferramentas, possam ser lidas pelo Editor E-DART. Em primeiro lugar, esta funcionalidade estaria disponibilizando versões E-DART para especificações E-LOTOS; em segundo lugar, estaríamos aumentando a quantidade de módulos disponíveis para reuso de código em novas especificações. Em suma, estaríamos permitindo que todas as vantagens do Editor E-DART pudessem ser aplicadas sobre uma especificação E-LOTOS, e não mais exclusivamente a especificações E-DART.

Integração com outras ferramentas

A gama de ferramentas existentes para verificação, geração de código fonte em linguagens de implementação, animação, etc. de especificações E-LOTOS esta aumentando. Fazer com que o Editor E-DART tivesse algum tipo de interação com estas ferramentas permitiria um melhor processo de criação dos sistemas, desde a especificação até a implementação.

Como geralmente estas ferramentas trabalham exclusivamente com código textual E-LOTOS o uso das mesmas sobre as especificações textuais geradas pelo tradutor E-DART/E-LOTOS do ambiente já é um fato interessante. Entretanto, esta pequena interação entre ferramentas ocorre em apenas um sentido, das especificações criadas no Editor E-DART para a análise das outras ferramentas. O caminho inverso, entretanto, também é interessante.

Imaginemos, por exemplo, a integração com uma ferramenta de animação de especificações. Normalmente o processo de animação envolve o destaque de ações da especificação indicando a sua ocorrência. Entretanto isso é feito sobre o texto E-LOTOS. O ideal seria se os elementos destacados fossem diagramas gráficos E-DART, o que melhoraria o processo de compreensão da animação.

Entre as implementações necessárias para que uma maior integração com outras ferramentas seja possível, está a criação de um modelo para definir *plug-ins* para o Editor E-DART. Estes *plug-ins* poderiam ser a própria ferramenta a ser integrada ao ambiente ou então um *gateway* para uma ferramenta já existente.

Exceções: Melhorando E-DART

Por fim, as novas versões do Editor E-DART deverão suportar também aspectos de E-LOTOS que não foram ainda "mapeados". Entre eles encontra-se o modelo de definição e tratamento de exceções introduzido em E-LOTOS. Um mecanismo gráfico deve ser capaz de tratar este assunto de forma clara e intuitiva para o usuário, visto que a versão textual não o faz com sucesso.

Anexo A - Manual do Usuário do Editor E-DART

Este documento apresenta as informações para a utilização do Editor E-DART. O manual é dividido em tópicos que descrevem os vários aspectos e funcionalidades do ambiente.

- A Interface Gráfica do Editor E-DART e as Funcionalidades do Ambiente
- HTTP: Construindo uma Especificação
- Salvando, Lendo e Reutilizando Módulos
- Customizando o Editor E-DART

Interface Gráfica e Funcionalidades do Ambiente

O Editor E-DART apresenta uma interface gráfica orientada a janelas que segue o padrão dos aplicativos Microsoft Office 97. Isto significa o emprego de barras de ferramentas para acesso às funções, menus flutuantes e sensíveis ao contexto. A maior parte dos elementos gráficos da interface podem ser customizados de acordo com as necessidades de cada usuário.

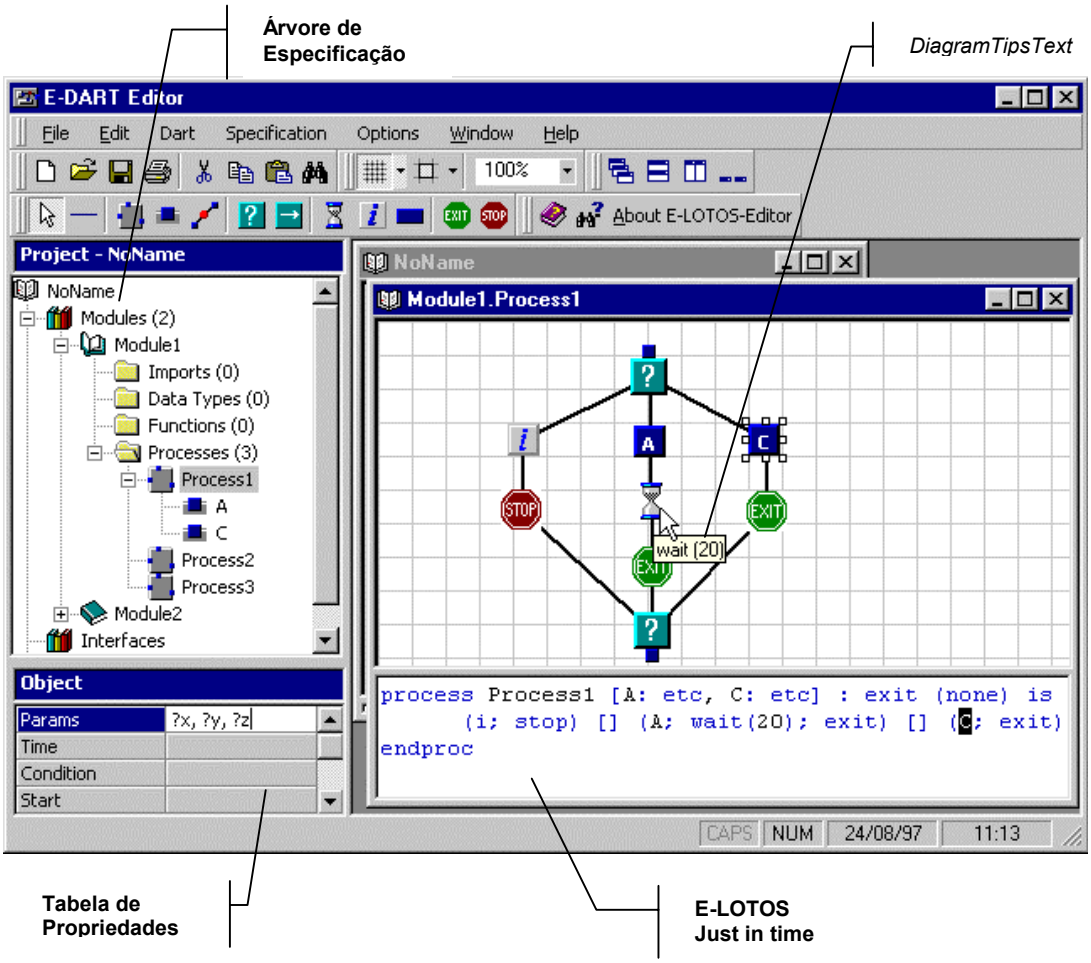


FIGURA A.1 - Interface gráfica do Editor E-DART

- **Árvore de Especificação**

A figura acima mostra os principais elementos da interface do Editor E-DART. A árvore de especificação é a funcionalidade mais importante na organização das especificações. A árvore organiza os sistemas em módulos. Cada módulo, por sua vez é organizado de acordo com quatro elementos principais: *Imports, Data Types, Function* e *Processes*.

Imports

Imports informa as referências externas de cada módulo, isto é, quais módulos externos são referenciados por um módulo em particular. As referências externas permitem que os processos de um módulos possam acessar funções, tipos de dados e processos definidos em outros módulos.

Uma referência é indicada por um nodo na pasta *Imports* com o nome do módulo referenciado. Para se criar uma referência devemos arrastar um módulo externo para a pasta de *Imports*. Isto por si só já define uma nova referência que então cria uma nova entrada na pasta para o módulo referenciado. Para eliminarmos uma referência precisamos apenas selecioná-la e pressionar a tecla *Del*. É importante lembrar que se eliminarmos uma referência a um módulo, os processos que antes usavam elementos do módulo eliminado podem tornar-se inconsistentes, pois a referência não mais existe.

Data Type e Function

Estas pastas contêm os conjuntos de tipos abstratos de dados e funções que cada módulo define. Cada função e tipo são identificados por uma entrada em suas respectivas pastas. Cada entrada é acompanhada do nome da função ou tipo definido. Para alterar este nome devemos acessar a tabela de propriedades do ambiente, explicada mais a frente.

Para criarmos um tipo ou função devemos acessar a função *Add Data Type* ou *Add Function*. Estas encontram-se no menu *Specification* do menu principal da ferramenta. Entretanto, a forma mais rápida de acessar as funções é através dos menus sensíveis ao contexto. Clicando-se com o botão direito sobre o nodo de um módulo podemos ver estas funções disponíveis no menu que aparece. Tanto *Add Data Type* quanto *Add Function* inserem um novo nodo na pasta correspondente. O nome do função ou tipo criados são gerados automaticamente pelo ambiente e podem ser posteriormente alterados.

Para definirmos o conteúdo de uma função um tipo basta clicarmos duas vezes sobre a entrada na árvore de especificação que os define. Isto abre uma janela texto comum, onde é possível, de forma textual, definirmos os campos de um tipo de dado, ou os procedimentos de uma função.

Como nas referências externas de *Imports*, uma função ou tipo podem ser excluídos de um módulos apenas pela sua seleção e posterior digitação da tecla *Del*.

Processes

A pasta *Processes* define o conjunto dos processos de um módulo. Deferentemente de *Imports, Data Type e Function*, cada nodo de *Processes* na verdade é outra sub-árvore para os vários elementos que compõe um processo.

A criação de uma nova entrada em *Processes* é feita de forma padrão. No menu *Specification* do menu principal do ambiente escolha *New Process* e uma nova entrada é criada. Novamente, um nome de processo é criado automaticamente pelo ambiente e pode ser alterado posteriormente. Para se excluir um processo basta selecioná-lo e teclar *Del*. A definição dos vários elementos que constituem um processo será mais tarde apresentada. Passamos agora para a apresentação dos outros elementos da interface gráfico do Editor E-DART.

- **Tabela de Propriedades**

A tabela de propriedades do Editor E-DART é a ferramenta que permite a parametrização de vários elementos de uma especificação. É através da tabela de propriedades que, por exemplo, definimos o nome dos módulos, processos e funções de uma especificação. O uso imediato desta ferramenta ser refere à alteração de valores para os elementos da sintaxe E-DART.

A tabela é dividida em várias linhas e duas colunas. A coluna da esquerda fornece o nome das propriedades que podem ser alteradas, enquanto que a coluna da direita apresenta os respectivos valores para estas propriedades. A alteração de um valor se dá por modificações na coluna da direita. Para isso basta clicar na entrada desejada e alterar o valor. As entradas na coluna da direita podem se comportar como caixas de edição, onde o usuário deve digital um valor, ou então como *ComboBoxes*, onde uma lista de valores possíveis é apresentada e o usuário escolhe um destes valores. Por fim, tem-se ainda um comportamento híbrido, onde a entrada fornece uma lista de valores possíveis, mas ainda assim o usuário pode digitar um valor que não se encontra na lista apresentada.

- **DiagramTipsText**

Em ferramentas modernas, quando o usuário repousa o mouse sobre um botão de interface por um determinado período de tempo, aparece próximo ao mouse uma pequena caixa de texto que descreve de forma muito resumida a funcionalidade daquele botão. Esta funcionalidade é chamada *ToolTipsText*, que poderia ser traduzida como "texto de dicas de ferramenta".

O Editor E-DART suporta esta funcionalidade de forma padrão para todos os elementos de interface que possuam alguma operação associada, caso principal dos botões das caixas de ferramentas. Além disso, o ambiente estende esta funcionalidade para os diagramas E-DART na janela de comportamento dos processos. Se o usuário repousar o mouse sobre um elemento E-DART, após alguns instantes surgirá próximo

ao mouse uma janela de texto que conterá uma porção de código E-LOTOS correspondente àquele elemento.

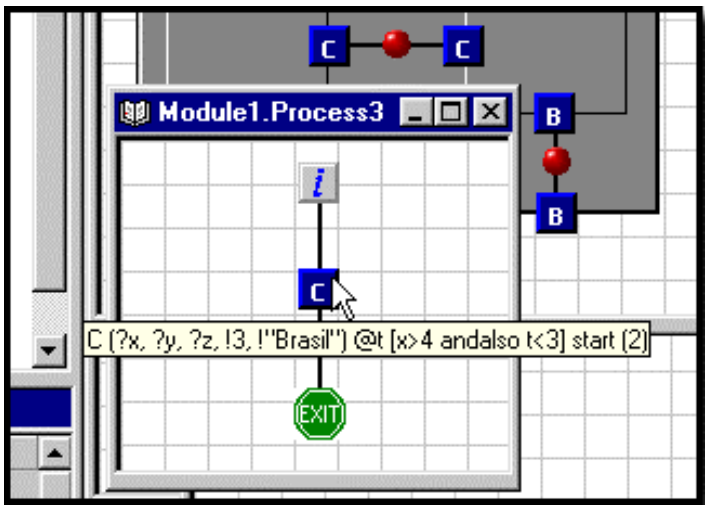


FIGURA A.2 - DiagramTipsText

A esta nova funcionalidade foi dado o nome de *DiagramTipsText*. Na figura acima podemos observar a ativação da funcionalidade sobre uma operação E-DART de sincronização sobre a porta C do processo *Process3*. O texto apresentado informa toda a construções E-LOTOS correspondente ao gráfico E-DART.

As *DiagramTipsText* são úteis quando estamos interessados em uma ação particular de um processo. Outro fato importante é a facilidade existente na obtenção do código E-LOTOS para este elemento por parte do usuário. Não existe navegação em menus convencionais ou sensíveis ao contexto. Não é necessário nem mesmo a seleção do elemento em que estamos interessados. Basta apenas apontá-lo e esperar por alguns instantes.

- **E-LOTOS Just in Time**

As *DiagramTipsText* apresentam código E-LOTOS para apenas um diagrama E-LOTOS específico. Entretanto, muitas vezes os usuários podem estar interessados em ter acesso ao código E-LOTOS de um processo inteiro. Isto é possível através da funcionalidade *E-LOTOS Just in Time*.

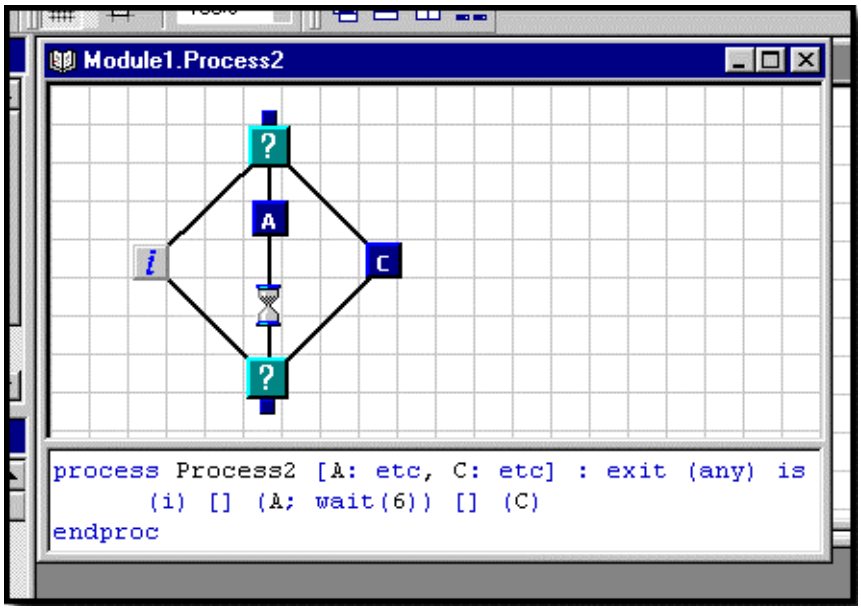


FIGURA A.3 - E-LOTOS Just in Time

Na parte inferior da janela de comportamento dos processos existe uma região que apresenta apenas texto, oculta por *default*. Se usuário desejar, esta área pode tornar-se visível e passa a apresentar o código E-LOTOS do processo correspondente. É interessante notar que a atualização deste código E-LOTOS é instantânea. Qualquer modificação na especificação que possa representar uma alteração em código E-LOTOS é imediatamente sentida na área textual. Isto garante que o código apresentado encontra-se sempre consistente em relação a especificação como um todo.

Manter o código E-LOTOS constantemente atualizado é possível porque o tradutor E-DART/E-LOTOS do ambiente é executado toda vez que uma alteração é realizada. O tradutor, quando requisitado atualiza todos os pontos modificados, o que garante a consistência permanente.

HTTP: Construindo uma Especificação

Nesta seção usaremos a especificação do protocolo HTTP para exemplificar o uso do Editor E-DART. A especificação trata o HTTP de uma forma bem abstrata mas na qual é possível exemplificarmos várias funcionalidades do ambiente.

- **Criando a Especificação**

O primeiro passo para a especificação do protocolo é criarmos no ambiente as primeiras definições. Assim, devemos inicialmente criarmos uma especificação E-DART através do comando *New Specification*. Para tal, acessamos o menu File do

menu principal do ambiente, ou o ícone correspondente na caixa de ferramentas superior.

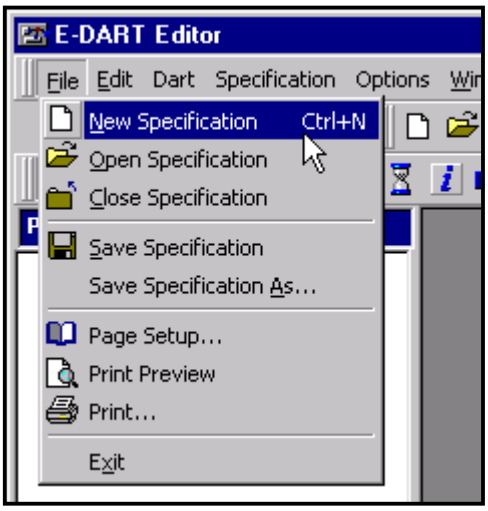


FIGURA A.4 - Menu File

A criação de uma nova especificação gera inicialmente uma estrutura na árvore de especificação. Temos assim a criação das pastas *Modules, Interfaces e Generic Modules*. A atual versão do Editor E-DART suporta manipulações apenas na pasta Modules.

FIGURA A.5 - Estrutura de uma especificação

Notamos também que é aberta uma janela gráfica no ambiente que corresponderá às ações iniciais tomadas na avaliação da especificação. Esta janela corresponderia, em uma analogia com a linguagem C, à função *main*.

Podemos agora dar um nome à nossa especificação. Isso é feito clicando-se duas vezes no *label* acima da árvore de especificação. Pode-se agora digital o novo nome que desejamos para nosso sistema, por exemplo "HTTP System".

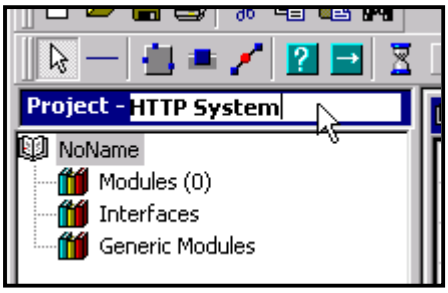


FIGURA A.6 - Alterando o nome de uma especificação

- **Módulos Cliente e Servidor**

O próximo passo da nossa especificação é a criação dos processos que definem o comportamento do protocolo. Como os processos só podem ser definidos dentro de módulos, primeiramente precisamos construir os módulos da especificação.

Cada módulos define um conjunto de processos, tipos abstratos de dados e funções, que podem ser posteriormente acessados e referenciados por outros módulos da especificação.

Vários modelos de organização de uma especificação podem ser empregados através dos módulos. No nosso caso teremos dois módulos principais que modelarão o comportamento de um cliente e de um servidor HTTP. Assim, o módulo *Client* definirá os processos, tipos de dados e funções que são relativos a um cliente HTTP. Da mesma forma ocorrerá com o módulo *Server*, referente ao servidor HTTP.

Para procedermos com a criação dos módulos deveremos acessar a função *New Module* do ambiente. Esta encontra-se no menu *Specification* do menu principal e também no menu sensível ao contexto ativado quando clicamos com o botão direto do mouse sobre a pasta Modules. O acesso às funções do ambiente sempre ocorre desta forma: através do menu principal da ferramenta, ou através de um menu sensível ao contexto. De agora em diante acessaremos as funções desejadas sempre através de menus sensíveis ao contexto, por estes serem mais rápidos. Entretanto, isso também pode ser feito pelo menu principal, como dito anteriormente.

Após a função *New Module* ser ativa, uma nova entrada na pasta Module é apresentada, correspondendo ao novo módulo criado. Tal módulo recebe um nome gerado automaticamente pelo ambiente que pode ser alterado. Para isso necessitamos selecionar o nodo do módulo e alterarmos a propriedade *Name* apresentada pela tabela de propriedades do ambiente.

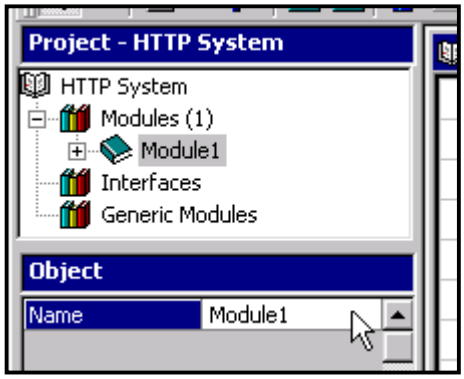


FIGURA A.7 - Alterando a propriedade *Name*

Notamos que a alteração desta propriedade automaticamente altera o nome da entrada correspondente ao nosso módulo na árvore de especificação. Ao primeiro módulos chamaremos de *Client*. Agora devemos proceder novamente com a criação de outro módulos e alterarmos seu nome para *Server*. Neste momento estamos prontos para iniciar a definição dos processos HTTP do nosso sistema.

- **Definição de Processos**

Iniciamos com a definição do processo cliente. Para tal acessamos a função *New Processos* clicando-se com o botão direito do mouse no módulo *Client*. Uma nova entrada é criada para o processo na pasta Processes do módulo. Alteremos agora o nome do processo para *Client* acessando a propriedade *Name* na tabela de propriedades.

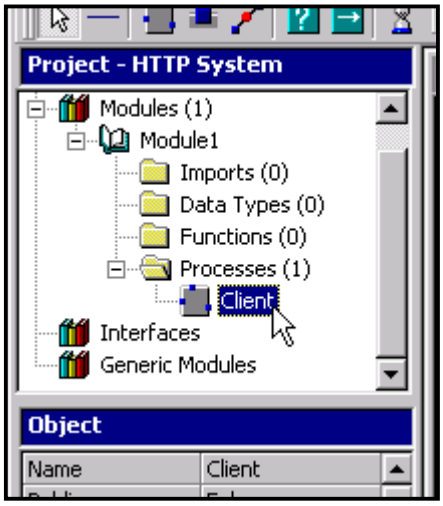


FIGURA A.8 - O processo *Client*

Nosso processo cliente deverá se comunicar com os outros processos através de uma porta de sincronização. Assim, devemos criar uma porta para o processo clicando-se, novamente com o botão direito do mouse, na entrada do processo e acessando o comando *New Port*. Uma entrada para a porta é criada e devemos alterar seu nome para S (de *Send*).

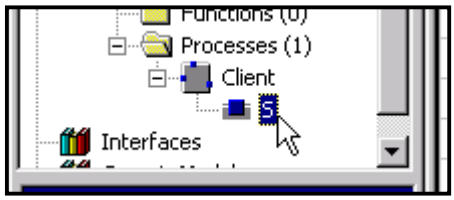


FIGURA A.9 - Porta de sincronização S

Neste momento temos definida a interface do processo *Client*, isto é, como o resto da especificação de nosso sistema "enxerga" o processo. A definição da interface do nosso processo cliente não define ainda seu comportamento, isto é, como o mesmo vai proceder quando executado. Isto será feito mais adiante, quando refinaremos mais nosso sistema.

Por enquanto devemos definir a interface agora do processo servidor HTTP. Para tal procedemos de forma semelhando ao cliente. Criamos um processo no módulo *Server*, damos a ele também o nome de *Server*, criamos uma porta de comunicação e damos o nome a ela de S. O nome da porta de comunicação não trata-se aqui de uma coincidência com a porta do processo cliente. Isso se deve ao fato de que a comunicação entre dois processos só pode ser possível através de portas de sincronização que possuam o mesmo nome.

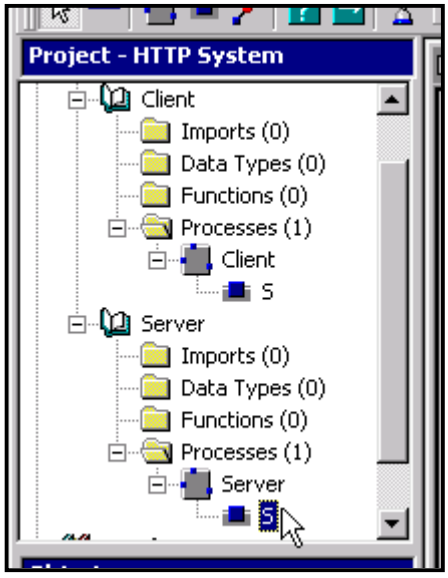


FIGURA A.10 - Árvore de especificação para os módulos *Client* e *Server*

Visto que temos agora definidas as interfaces dos processos cliente e servidor, podemos definir como são as relações entre estes processos. Para fazermos isso usaremos agora a janela de especificação. Clicando duas vezes no nodo raiz da árvore de especificação forçaremos que a janela correspondente venha à frente.

Primeiramente devemos incluir nesta janela um retângulo de instanciação de processos. Parra isso acessamos a função *Process Instancing* e definimos, com o mouse, um retângulo na janela de especificação. O retângulo gerado inicialmente não possui nenhuma processo interno sendo instanciado.

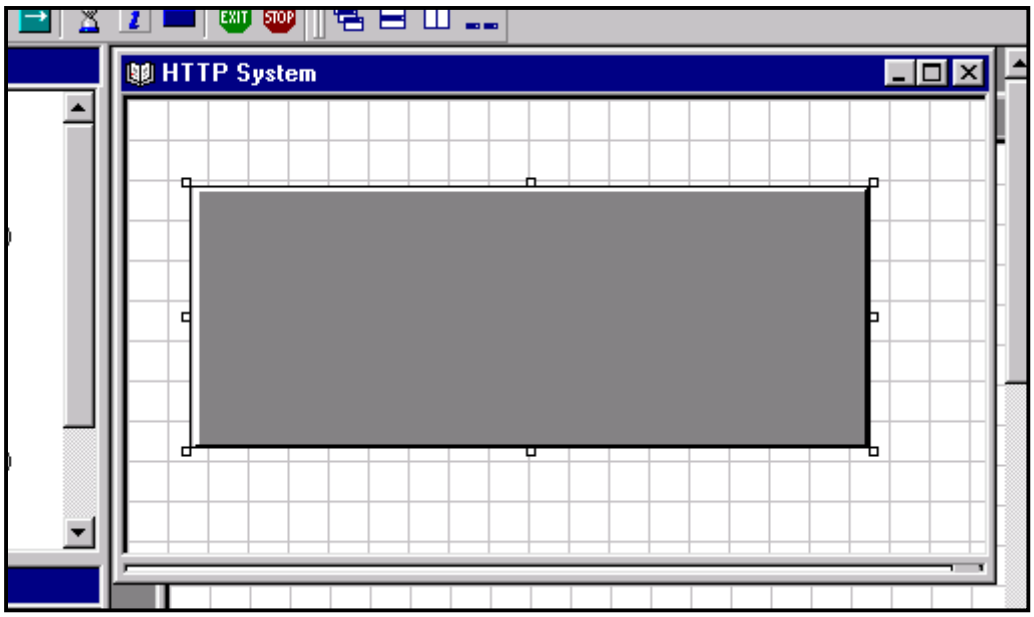


FIGURA A.11 - Retângulo de instanciação de processos

O próximo passo é instanciarmos os processos cliente e servidor HTTP. Antes, devemos alterar a propriedade *Public* dos processos *Client* e *Server* para permitir que os mesmos possam ser utilizados fora de seus módulos. Depois, devemos arrastar uma referência desses processos para o retângulo de instanciação criado anteriormente. Assim, devemos clicar, na árvore de especificação, no nodo do processo que estamos instanciando e arrastá-lo até a janela de especificação.

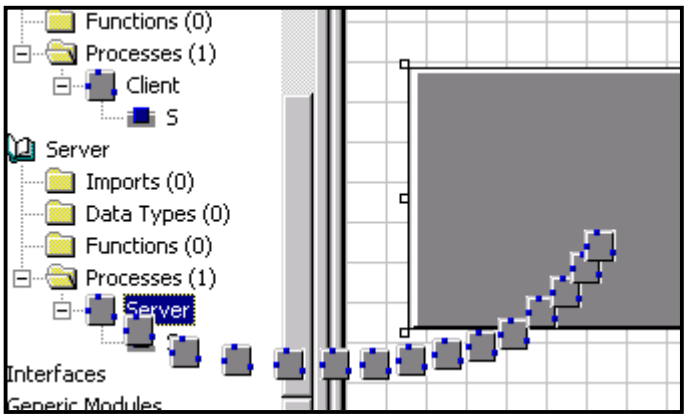


FIGURA A.12 - Instanciando o processo *Server*

Nosso retângulo de instanciação agora define a instanciação de um processo *Server* e de um processo *Client*. Agora devemos definir como estes processos instanciados se relacionam, isto é, como ocorre a troca de informações entre os processos. Usamos nessa definição a porta de comunicação S definida na interface dos processos. Ela será o meio pelo qual os processos se comunicarão.

Assim, devemos solicitar a função *Add Conection* que permite a definição de uma comunicação entre duas portas de mesmo nome de processos distintos.

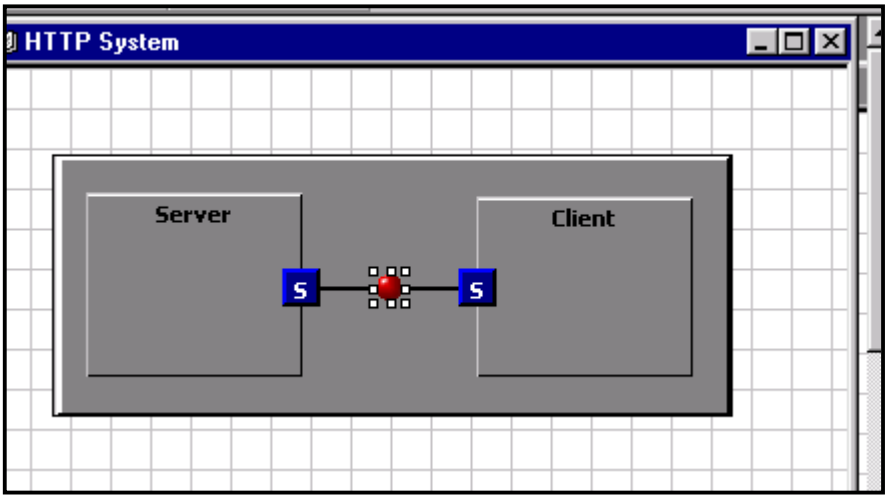


FIGURA A.13 - Conexão entre *Client* e *Server*

Até este momento temos definidos os módulos, processos e interações entre os processos de nosso sistema. Já temos uma idéia abstrata de como o mesmo funciona. Dois processos distintos, um servidor e um cliente, trocam informações através de uma porta de comunicação S. Agora queremos definir o comportamento de nosso sistema de uma forma mais detalhada. Para isso devemos definir o que efetivamente os processos *Client* e *Server* fazem internamente, isto é, queremos definir seu comportamento dinâmico.

- **Comportamento Dinâmico dos Processos**

Para a definição do comportamento dinâmico dos processos interagimos em grande parte com a janela de especificação de cada processo. É nesta janela onde efetivamente informamos quais ações devem ser executadas e em que tempo as mesmas devem ser avaliadas. Com isso temos um conjunto de gráficos que representam o modo como um processo evolui e quanto o mesmo interfere no ambiente em que está inserido. A interferência no ambiente de um processo se dá quando executamos uma ação de sincronização, onde informações podem ser passadas a outro processos através das portas de sincronização definidas na interface do processo.

Inicialmente começamos pela definição do comportamento de Client. Novamente, devemos clicar duas vezes na entrado do processo na árvore de especificação para que a janela de especificação do processo torne-se ativa. Os procedimentos que o cliente HTTP deve tomar são os seguintes:

1. Construir uma mensagem de requisição de uma páginas HTML;
2. Enviar o pedido construído ao servidor HTTP através da porta S;
3. Esperar na porta S por uma mensagem de retorno, por um período de tempo máximo determinado;
4. Extrair da mensagem de retorno a páginas HTML requisitada;
5. Mostrar a páginas de reposta da operação em um Browser.

Algumas ações envolvidas nos procedimentos acima podem ser abstraídas. Por exemplo, os procedimentos de construção de uma mensagem, recuperação de uma páginas HTML e a exibição desta páginas podem ser apenas referenciados sem serem definidos. Para que isso fosse feito necessitaríamos de outros níveis de abstração do sistema. A título de exemplo manteremos estes procedimentos como sendo já definidos e abstrairemos sua avaliação.

A construção de uma mensagem de requisição de uma páginas HTML exige a existência de alguns outros elementos no módulo *Client*. Assim, necessitamos do tipo *HTTP_Message* que define uma mensagem HTTP, e da função *Built_Request* que retorno um mensagem HTTP correspondente a um pedido de página HTML. Por fim, no processo *Client* precisamos ainda de uma variável local do tipo *HTTP_Message* responsável por armazenar as requisições construídas por *Built_Resquest*.

Para criamos um tipo de dados (*HTTP_Message*) devemos acessar a função *Add Type*. Para criarmos uma nova função chamamos *Add Function,* já mencionadas anteriormente neste manual. Para criarmos uma variável local de um processo clicamos com o botão direto no processo desejado e escolhemos a função *Add Local Var*. Novamente podemos alterar o nome da variável a pouco criada alterando o valo da propriedade *Name* da variável (neste caso chamaremos a variável de *Resquest*).

Agora devemos definir da janela de especificação a primeira ação do nosso processo cliente. Devemos atribuir à *Request* o valor de retorno de *Built_Request*. Isso é feito logicamente com o operador de atribuição de E-DART. Para inserir este operador na janela de especificação selecionamos o ícone correspondente, na caixa de ferramentas do E-DART do ambiente, e logo em seguida delimitamos um retângula da janela.

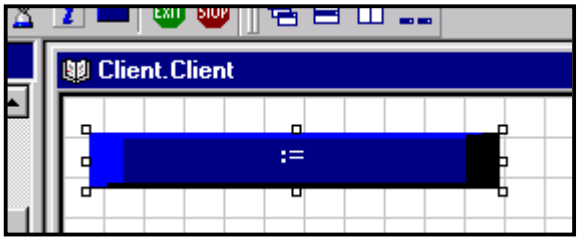


FIGURA A.14 - Atribuição

Devemos observar que o operador possui internamente apenas o símbolo :=, indicando que nenhum dos lados da atribuição foram definidos. Para fazê-lo devemos alteras as propriedades correspondentes na tabela de propriedades da atribuição.

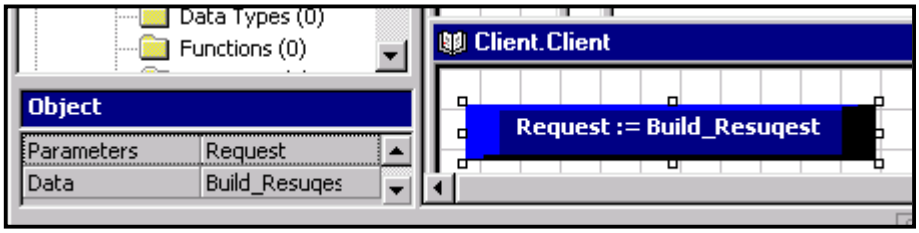


FIGURA A.15 - Parâmetros da atribuição

A segunda ação em *Client* é enviarmos a mensagem HTTP contida em *Request* ao servidor, através da porta S. Isso é feito com a ação de sincronização. A

definição desta ação é feita arrastando-se, da árvore de especificação, a porta sobre a qual o processo deve sincronizar, neste caso S. Isso inclui na janela de especificação um quadrado semelhante a uma porta de sincronização, que na verdade denota uma sincronização sobre a porta indicada.

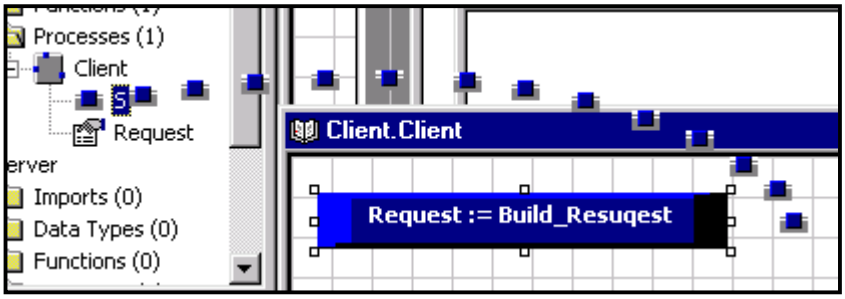


FIGURA A.16 - Definindo uma sincronização

Agora devemos preencher os vários parâmetros envolvidos na sincronização através da tabela de propriedades.

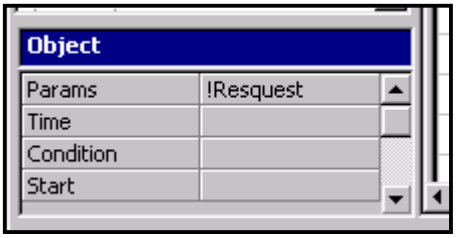


FIGURA A.17 - Parâmetros da primeira sincronização

A próxima ação executada também é uma sincronização em S, só que desta vez para esperarmos uma resposta do servidor HTTP. Nesta segunda sincronização usaremos uma nova variável local *Reply*, também do tipo *HTTP_Message*. Neste caso teremos a seguinte tabela de propriedades para a segunda sincronização.

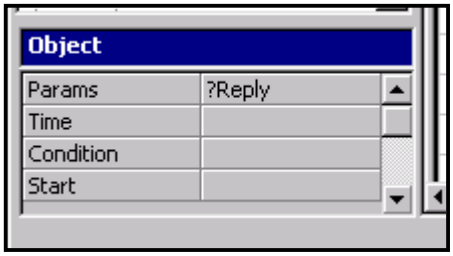


FIGURA A.18 - Parâmetros da segunda sincronização

O penúltimo passo tomado é extrairmos da mensagem HTTP de resposta, vinda do servidor, uma página HTML para ser mostrada por um Browser. Neste caso necessitamos de mais um tipo de dados, *HTML_Page*, e uma nova variável local *Page*. Ainda necessitamos da função *Extract_HTML_Page* que retira de uma mensagem HTTP uma páginas HTML, se esta existir.

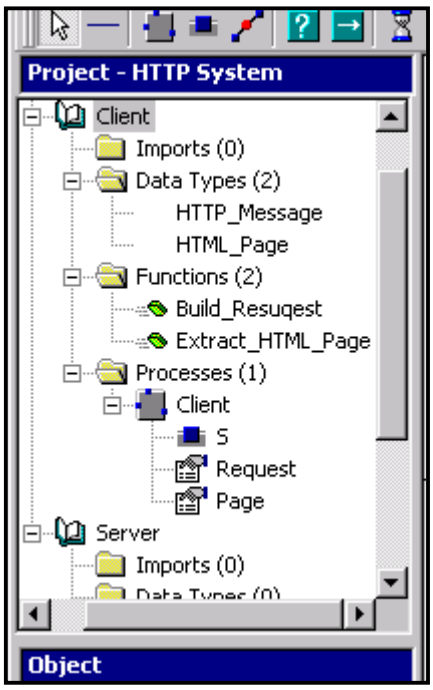


FIGURA A.19 - Árvore de especificação com funções, tipos de dados e variáveis locais

Assim, podemos fazer mais uma atribuição final: *Page* receberá o retorno de *Extract_HTML_Page*, que recebe como parâmetro a mensagem HTTP armazenada em *Reply*.

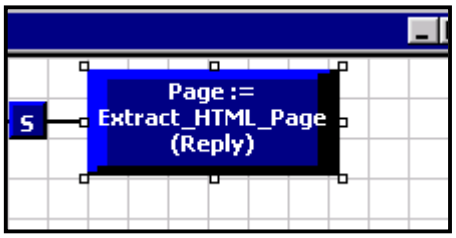


FIGURA A.20 - Última atribuição do processo *Client*

E finalmente necessitamos agora apenas apresentar a páginas HTML retornada pelo servidor em um Browser. Evidentemente que esta ação envolve a existência de um comportamento. Assim, criamos no módulo *Sender* um novo processo *ShowInBrowser*, que abstrai a apresentação de uma página HTML.

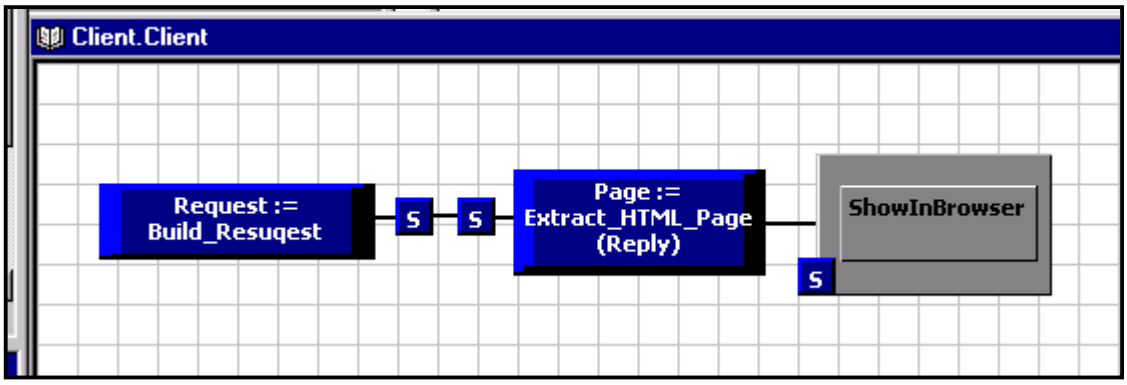


FIGURA A.21 - O processo *Client*

Desta forma temos definido o comportamento do processo cliente e expandido, de acordo com as necessidades, os elementos do módulo *Client*. Isso novamente deve acontecer com *Server* para que o processo servidor possa ser corretamente especificado.

Salvando, Lendo e Reutilizando Especificações

Anteriormente exemplificamos o uso do Editor E-DART na construção de um sistema cliente/servidor para o protocolo HTTP. Naquela especificação foram criados dois módulos com alguns tipos de dados, funções e processos cada um. Descreveremos aqui como proceder com o salvamento, leitura e reuso de especificações através da utilização do mecanismo de modularização da ferramenta.

As especificações criadas podem ser obviamente armazenadas em arquivos para sua posterior recuperação. Existe, entretanto, duas formas distintas para o armazenamento das especificações. Estas formas distintas estão relacionadas com os módulos dos sistemas.

- **Salvamento e Leitura Triviais**

A primeira forma de armazenamento é trivial. Uma especificação é salva em um único arquivo que conterá todos os módulos do sistema bem como seus processos e outros elementos. A recuperação da especificação para esta forma de armazenamento também é trivial. Lendo-se o arquivo a especificação é novamente construída pela ferramenta permitindo ao usuário alterar os dados gravados.

Para acessar as funções de salvamento e leitura devemos usar o menu principal do ambiente. No menu *File* escolher *Save, Save As... ou Open*.

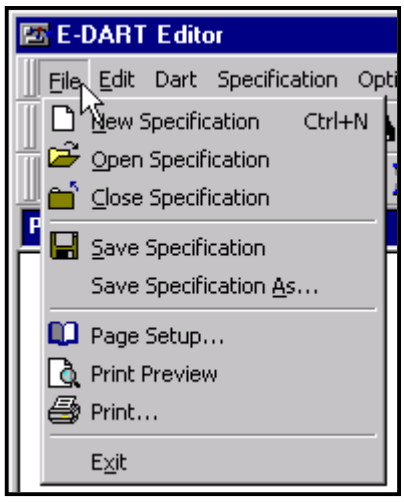


FIGURA A.22 - As funções do menu File

- **Salvamento e Leitura de Acordo com os Módulos**

A segunda forma de armazenamento, por outro lado, leva em conta o fator modularização. Nesta forma os módulos de um sistema podem ser armazenados de forma independente em vários arquivos separados. Para cada módulo de uma especificação temos um arquivo correspondente. A especificação como um todo também possuirá um arquivo correspondente. Entretanto, este arquivo possuirá pouca informação, mas basicamente o conteúdo mais importante são os dados relativos aos módulos que constituem a especificação. Tem-se apenas referências a estes módulos, mais especificamente aos arquivos que armazenam os módulos.

A recuperação de uma especificação armazenada com módulos independentes é um pouco mais complexa. Primeiro é lido o arquivo principal que organiza a especificação e referencia os módulos constituintes. Para cada um destes módulos tem-se então a leitura do arquivo correspondente. Ao final, as referências internas da especificação são refeitas e a especificação reconstituída.

Para salvarmos módulos de forma independente usa-se a função *Save Module* do menu sensível ao contexto ativado clicando-se com o botão direito do mouse sobre o módulo desejado.

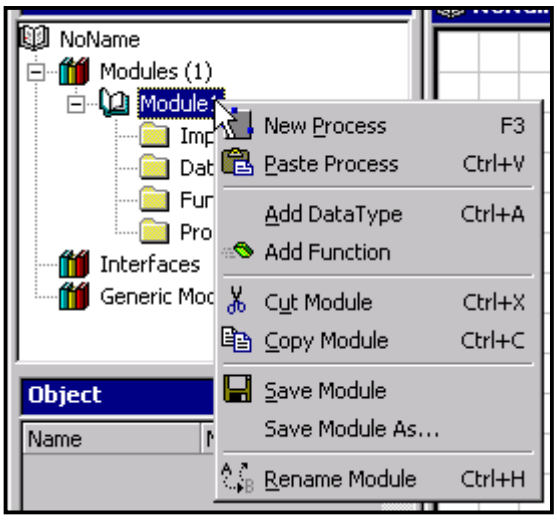


FIGURA A.23 - Menu sensível ao contexto para um módulo

Outra forma, mais automática, é marcarmos a opção *Generate separed module files* da janela de propriedades da especificação. Para acessarmos esta janela devemos selecionar em *Specification*, do menu principal, a opção *Properties*. Marcar esta opção força que toda vez que a especificação for gravada os módulos da mesma serão armazenados de forma independente.

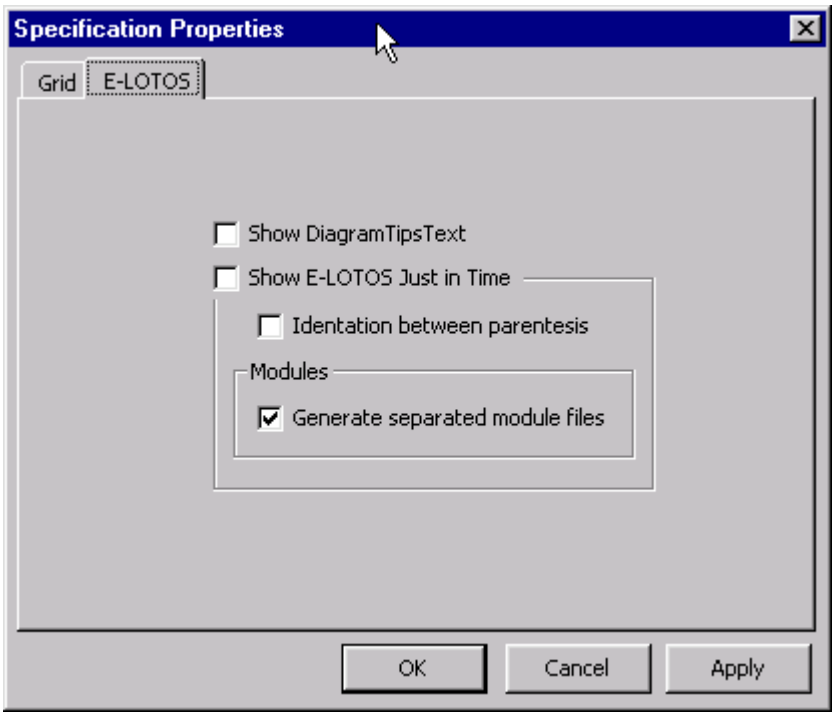


FIGURA A.24 - Janela de propriedades

- **Reutilizando Módulos em outras Especificações**

Obviamente a fato de termos duas formas distintas de armazenamento tem sua razão de existir. Com o armazenamento orientado à módulos a ferramenta implementa

a funcionalidade de reuso de módulos. Isto porque é possível importar um módulo específico para dentro de uma especificação. Como os módulos podem ser armazenados independentemente, eles podem também ser importados independentemente. Assim, uma nova especificação pode ser construída através do reuso de vários outros módulos definidos anteriormente por outras especificações.

Para importarmos um módulo para dentro de nossa especificação devemos acionar a função *Add Module*. Isso pode ser feito através do menu principal em *Specification*, ou então através do menu sensível ao contexto da pasta Modules da especificação.

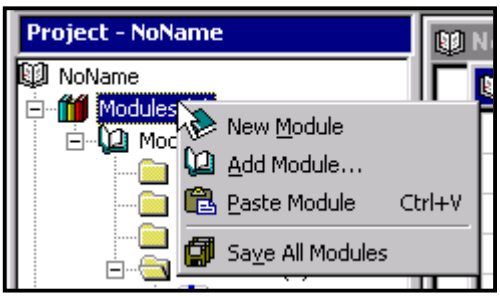


FIGURA A.25 - Menu sensível ao contexto para os módulos

Além da vantagem óbvia do menor tempo de desenvolvimento, o reuso de módulos traz mais segurança às novas especificações. Um módulo construído e testado de forma correta por uma especificação não precisa ser novamente testado pelas especificações que venham a usar este mesmo módulo.

- **Salvando Especificações como E-LOTOS**

As *DiagramTipsText* geram E-LOTOS no nível de elementos específicos. *E-LOTOS Just in Time*, respondem por código E-LOTOS a nível de processos. Por fim, se quisermos código E-LOTOS para toda uma especificação devemos utilizar esta última funcionalidade: o armazenamento E-LOTOS.

Quando executamos a função de salvamento de uma especificação os arquivos gerados obedecem um padrão interno do ambiente de armazenamento dos dados. Este padrão garante a posterior recuperação das especificações criadas. Entretanto, podemos escolher, na momento do salvamento, a forma de armazenamento desejado: ou especificações E-DART (*.spc) ou código E-LOTOS (*.els).

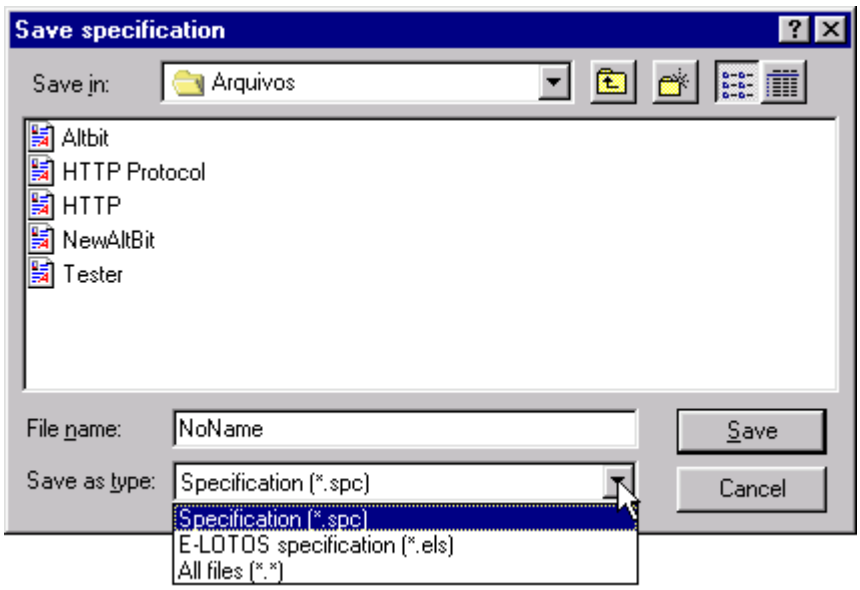


FIGURA A.26 - Opções de salvamento de uma especificação

O armazenamento E-LOTOS gera arquivos texto que contém toda a especificação E-DART em texto E-LOTOS. Estes arquivos podem então ser utilizados por outras ferramentas para validação, teste ou animação da especificação. O único inconveniente no armazenamento E-LOTOS é o fato de que o Editor E-DART ainda não é capaz de construir gráficos E-DART a partir de fontes textuais. Ainda que isto fosse possível, a reconstituição E-DART não seria perfeita, pois em um arquivo E-LOTOS não são armazenadas informações gráficas como posicionamento e dimensões dos elementos E-DART.

Customizando o Editor E-DART

O Editor E-DART pode ter seu ambiente customizado de várias formas. Primeiramente podemos escolher quais comandos estarão presentes nas barras de ferramentas do ambiente. Isso é feito clicando-se com o botão direito do mouse sobre a área onde encontram-se as barras de ferramentas - tipicamente no alto da janela do editor. Um menu sensível ao contexto aparecerá e deve-se escolher então a opção Customize.

Esta customização permite que se indique quais comandos estarão em que barra de ferramentas e quais barras de ferramentas estarão visíveis no ambiente. As barras pré-configuradas são as seguintes:

- *Standard* - Apresenta funções padrão de manipulação de arquivos, impressão e de copy & paste;
- *E-DART* - Apresenta funções para a criação dos elementos E-DART na janela de especificação dos processos;

- *Frame* - Apresenta funções para manipulação de propriedades de zoom e grid das janelas de especificação dos processos;
- *Windows* - Apresenta funções para manipulação das janelas MDI do ambiente, como maximização, organização de ícones e de tile horizontal e vertical; e
- *Help* - Apresenta funções para acesso aos arquivos de ajuda do ambiente.

Outro tipo de customização permitida é em relação a apresentação das especificações. Esta customização é feita através da janela de propriedades do ambiente, que é acessada através do menu *Specification* do menu principal.

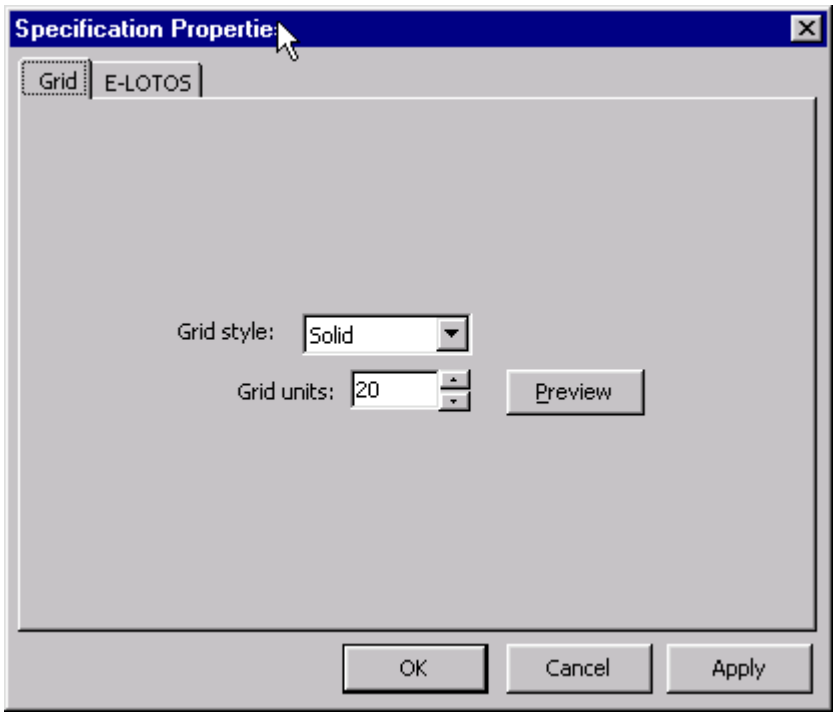


FIGURA A.27 - Configuração do Grid

- **Grid**

Na seção *Grid* temos a customização das linhas de grid das janelas de especificação. As alterações feitas aqui serão aplicadas sobre todas as janelas, abertas ou não. A tecla *Preview* permite uma visualização prévia das alterações realizadas.

Grid style: Define como serão impressas as linhas de grid. O valor default é *Solid* que imprime uma linha contínua sem interrupções. Outras opções permitem linhas tracejadas, pontilhadas, tracejadas e pontilhadas, entre outras.

Grid units: Define o número de pontos que separa um alinha de grid de outra. Estes pontos correspondem tanto a distância entre linhas horizontais quanto verticais.

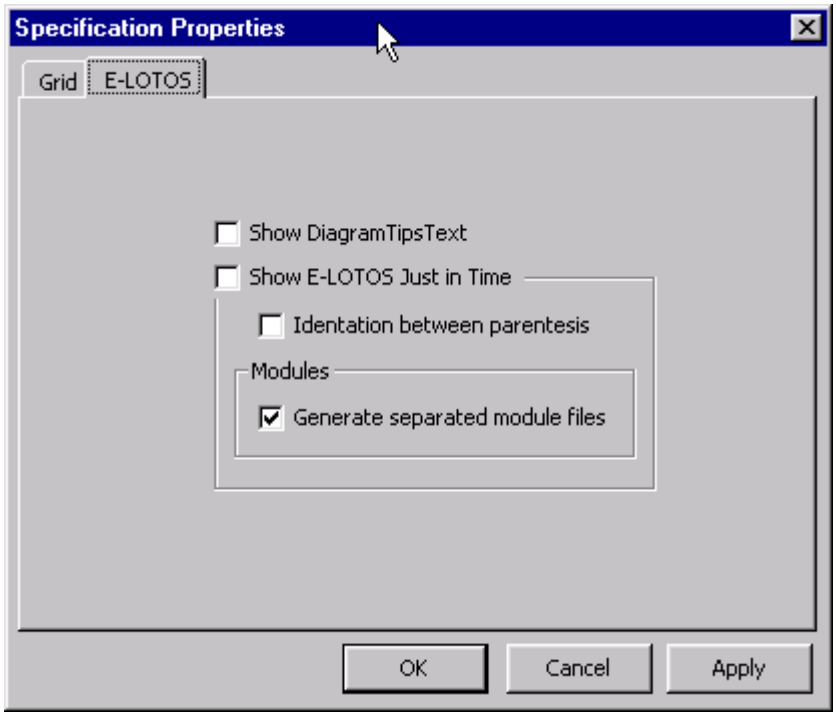


FIGURA A.28 - Configuração de parâmetro E-LOTOS

- **E-LOTOS**

A seção E-LOTOS é responsável pela customização na geração automática de código E-LOTOS para os diagramas E-DART das especificações.

Show DiagramTipsText: Indica se as *DiagramsTipsText* devem ser apresentadas ao usuário toda vez que o mesmo repousar o mouse sobre um elemento E-DART por alguns instantes.

Show E-LOTOS Just in Time: Indica se o código E-LOTOS para todo um processo E-DART deve estar visível ao usuário, na parte inferior da janela de especificação dos processos.

Identation between parantesis: Indica como deve ocorrer a indentação de expressões E-LOTOS, em *E-LOTOS Just in Time*, quando tem-se a necessidade de geração automática de parêntesis.

Generate separated module files: Indica se os módulos dos sistemas devem ser armazenados em arquivos independentes.

Anexo B - Documentação sobre a Implementação do Editor E-DART

Este documento tem como objetivo apresentar os mais importantes detalhes técnicos envolvidos na implementação do Editor E-DART. A documentação refere-se principalmente ao uso e desenvolvimento de controles ActiveX necessários para formar a interface gráfica do ambiente, bem como em relação a estrutura de classes criada para suportar a representação interna de uma especificação E-DART.

Ambiente de desenvolvimento

O Editor E-DART foi escrito no ambiente Microsoft Visual Basic 5.0. Este foi escolhido por ser de fácil compreensão, possuir o melhor suporte aos componentes ActiveX, extensamente utilizados na ferramenta, além de apresentar o melhor suporte a criação de novos componentes.

A seguir são apresentados os componentes ActiveX utilizados na ferramenta, bem como explicada a necessidade de uso de cada um deles.

Componentes ActiveX

Como dito anteriormente os componentes ActiveX foram escolhidos pela grande variedade disponibilizada atualmente por terceiros. Existem componentes para suportar diversos aspectos envolvidos na criação de sistemas. No caso do Editor E-DART, os componentes ActiveX foram extensivamente utilizados na formação da interface gráfica do sistema. A seguir descreveremos quais componentes foram utilizados para implementar quais funcionalidades.

- **Delimitador de Área**

Um delimitador de área se fez necessário para que pudéssemos deixar a cargo do usuário a decisão de como melhor disponibilizar as funcionalidades presentes na interface gráfica do ambiente. Basicamente um delimitador divide uma área retangular em duas partes. Esta divisão, por sua vez, pode ser alterada a qualquer momento pela intervenção do usuário.

Um exemplo no uso de delimitadores é na definição dos espaços ocupados pela árvore de especificação e pela tabela de propriedades. Existe um delimitador entre estas funcionalidades que define como elas serão dispostas.

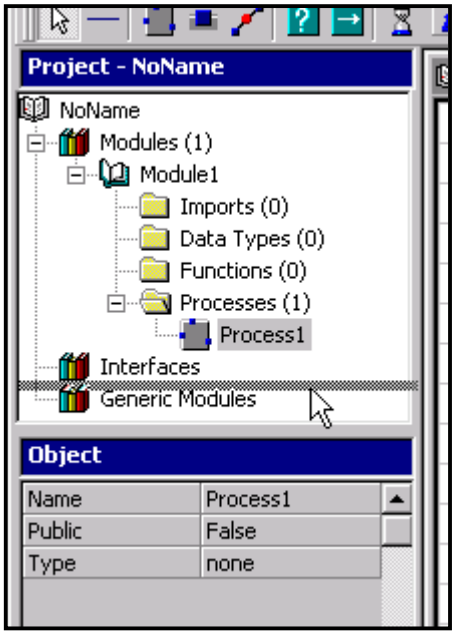


FIGURA B.1 - Delimitador para árvore de especificação e tabela de propriedades

Outro exemplo é o delimitador usado na apresentação do código E-LOTOS da funcionalidade *E-LOTOS Just in Time*. Junto a janela de especificação de cada processo existe, na parte inferior, um delimitador que "esconde" o código E-LOTOS. Movendo-se tal delimitador mais para cima podemos visualizar então tal código.

A implementação da funcionalidade fornecida pelo delimitar se deu através do uso do controle *Splitter* do pacote *Sheridam Active Threed* [SHE 97]. O controle divide uma área em duas onde é possível determinar-se quais outros controles estarão presentes. Em tempo de execução, a divisão de área promovida pelo controle pode ser alterada livremente pelo usuário.

- **Menus padrão Microsoft Office 97**

A última versão do pacote Office da Microsoft apresenta um esquema de menus muito interessantes. Originários inicialmente dos menus do Internet Explorer 3.0, os menus do Office 97 possuem um alto apelo visual com o emprego do chamado *Flat-Look,* onde os botões existentes tem suas bordas visíveis apenas quando o mouse estiver sobre eles.

O aspecto visual é muito importante porque fornece ao usuário uma interface mais agradável e profissional. Mas mais importante que isto são as funcionalidade que os menus do Office oferecem. Entre as principais encontram-se:

- Disposição dos menus e caixas de ferramentas em qualquer um dos lados da janela do ambiente, em então sendo mostrados como janelas flutuantes. Isso permite ao usuário disponibilizar os menus e caixas de forma que melhor lhe convier no ambiente.

- Customização das caixas de ferramentas. As caixas de ferramentas nada mais são que conjuntos de botões que acessam funções de forma mais imediata no ambiente. Com os menus do Office 97, pode-se determinar quais funções estarão presentes em que caixa de ferramenta. Pode-se até mesmo identificar quais caixas estarão visíveis ao usuário.
- As caixas de ferramentas, além de conterem botões que são "atalhos" para funções do ambiente, podem ainda conter listas de valores, listas de strings, entre outras. Isso permite por exemplo que o valor de zoom de uma janela de especificação possa ser escolhido entre um conjunto de valores.

Para a implementação dos menus no estilo do Office 97 foi utilizado o controle ActiveX *ActiveBar* da *Data Dynamics* [DAT 97]. Junto ao controle existe um editor de ícones para a criação dos botões de atalho e um editor de caixas de diálogo, que permite customizar as caixas e menus *default* do sistema.

- **Árvore de Especificação**

Para a árvore de especificação necessitávamos de um controle capaz de gerenciar estruturas de nodos com sub-nodos. Deveríamos ser capaz de expandir um nodo ou então fechá-lo.

Entre os controles comuns da API do Windows 95 e NT 4.0, foi introduzido um controle que gera este tipo de estrutura. Entretanto o uso do controle através da API do Windows é uma tarefa muito complexa, além de não estar de acordo com o estilo de construção dos sistemas do Visual Basic.

Entretanto, no próprio pacote do Visual Basic encontramos o controle *TreeView*, desenvolvido pela Microsoft, que fornece uma interface com a API do Windows, o que permite que utilizemos o controle de forma conveniente no Visual Basic.

- **Tabela de Propriedades**

A funcionalidade da tabela de propriedades é necessária para a edição de parâmetros dos vários elementos E-DART. A tabela de propriedades é dividida em duas colunas e várias linhas; uma linha por propriedade. A coluna da esquerda deve possuir o nome das propriedades, enquanto que a coluna da direita deve possuir o valor para cada uma das propriedades da coluna da esquerda.

Apenas a coluna da direita pode ser editada. Clicando-se em uma entrada da coluna da esquerda o procedimento deve ser o de ativar a edição do valor da propriedade na coluna da direita.

Além disso, a coluna da direita deve possuir três tipos distintos de edição. O primeiro refere-se à edição comum de propriedades, onde o usuário deve apenas clicar na entrada e digitar um novo valor para a propriedade. O segundo tipo se refere a uma lista de valores. Quando uma entrada for acessada, um símbolo representado uma lista

deve surgir na coluna de direita. Clicando-se neste símbolo um lista de valores aparecerá e o usuário poderá escolher um destes valores. Por fim, o último tipo de edição trata-se também de uma lista de valores, mas que não obriga o usuário de escolher um valor da lista, deixando-o livre para que possa também digitar um valor diferente.

Para implementar a tabela de propriedades usamos o controle *SSDBGrid* do pacote *Sheridan Data Widgets* [SHE 96], que era o único que fornecia as funcionalidades descritas acima na edição de um campo da tabela.

- **Texto para *E-LOTOS Just in Time***

A funcionalidade *E-LOTOS Just in Time* deve apresentar de forma textual um código E-LOTOS para um processo E-DART. A apresentação deste código deve ser clara, visto que códigos E-LOTOS são mais difíceis de serem compreendidos.

Para tornar essa visualização mais clara, optou-se por usar cores distintas para as palavras reservadas a linguagem E-LOTOS. Assim, necessitávamos de um controle onde pudéssemos controlar a cor de cada palavra que seria impressa. Além disso, o controle deveria possuir a capacidade de imprimir o código com diferentes fontes Windows, com diferentes tamanhos.

Assim, foi utilizado o controle *RichTextEdit*, também desenvolvido pela Microsoft.

- **Gráficos E-DART**

Obviamente necessitávamos de um controle que gerenciasse os gráficos E-DART de forma conveniente. Tal controle deveria possuir a capacidade de conectar elementos, além de estabelecer relações entre os mesmos; possuir funcionalidade para seleção múltipla, movimentação e troca das dimensões dos gráficos; zoom e grid.

Infelizmente este tipo de controle não é muito comum e os poucos controles analisados mostraram-se ineficientes na disponibilização de todas as funcionalidades de manipulação de gráficos necessárias. Desta forma, a solução foi a criação de um novo controle, que demos o nome de *DragDropWindow*, que atendesse as exigências dos gráficos E-DART.

Com este conjunto de componentes ActiveX foi possível a implementação de todas as funcionalidades definidas para o ambiente do Editor E-DART. A seguir apresentaremos outras implementações usadas na construção da ferramenta.

Representação Interna de uma Especificação

A representação interna de uma especificação se refere às estruturas criadas para manter e gerenciar uma especificação em memória. Cada elemento E-DART de uma especificação constitui-se em um objeto que é acessado através de suas propriedades e métodos.

Os objetos E-DART podem possuir ou não uma representação gráfica. Aqueles que possui uma representação gráfica devem referenciar também outro objeto que define tal representação. Um exemplo deste tipo de objeto é um operador de *delay*, onde uma ampulheta na janela de especificação é sua representação gráfica. Um elemento E-DART como um módulo, por exemplo, não possui representação gráfica. Desta forma não necessita referenciar um objeto gráfico correspondentes.

Os objetos gráficos de uma especificação são todos da classe *ShapeObj*, exportada pelo controle *DragDropWindow*. Cada objeto *ShapeObj* possui os seguintes métodos e propriedades públicos.

Propriedades de *ShapeObj*

- UIDName – Identificador único
- Picture - Imagem gráfica do objeto
- Caption – Texto apresentado junto à imagem do objeto
- Font – Fonte usada no texto do objeto
- ForeColor – Cor usada para o texto do objeto
- ToolTipText – Texto a ser apresentado quando o mouse repousar repousar
- Visible – Indica se o objeto está visível ao usuário
- Tag – Valor genérico usado para várias operações
- Sources – Referência a todos os objeto gráficos que precedem
- Dests – Referência a todos os objetos gráficos que sucedem
- ShowDragging – Indica se haverá apresentação de gráficos quando o objeto estiver sendo movido
- AllowSelect – Indica se o objeto pode ser selecionado pelo usuário
- AllowResize – Indica se o objeto pode ser redimensionado pelo usuário
- InsideShapes – Referência a todos os objetos gráficos internos
- ParentShape – Referência ao objeto gráfico ao qual o objeto pertence
- DragDropParentWindow – Janela gráfica ao qual o objeto pertence
- Selected – Indica se o objeto encontra-se selecionado
- Top, Left – Posição do objeto na janela gráfica
- Width, Height – Dimensões do objeto

Métodos de *ShapeObj*

- AddShape – Adiciona um novo objeto gráfico aos objetos internos
- RemoveShape – Remove um objeto gráfico dos objetos internos
- Refresh – Atualiza graficamente o objeto
- Move – Altera a posição e dimensões do objeto
- Zorder – Trás o objeto para a frente dos outros objetos ou envia para trás
- AddSource – Adiciona um objeto gráfico ao conjunto de precedentes
- AddDest – Adiciona um objeto gráfico ao conjunto de sucessores

Com as funcionalidades dos objeto *ShapeObj* os elementos gráficos E-DART podem ser representados eficientemente nas janelas de especificação dos processos. Devemos ressaltar que objetos *ShapeObj* estão intimamente relacionados com outros objeto *ShapeObj* e com objetos *DragDropWindow*. Outros objetos *ShapeObj* constituem objetos que precedem um sucedem o objeto original na janela gráfica. Isso forma uma seqüência gráfica de objetos que define a sucessão de ações E-DART.

DragDropWindow é o objeto responsável pela implementação da janela gráfica dos processos, e contém os objetos que representam graficamente os elementos E-DART. Sobre uma janela gráfica podemos aplicar várias operações, e vários parâmetros podem ser alterados. Apresentamos agora os principais métodos e propriedades de um objeto *DragDropWindow.*

Propriedades de *DragDropWindow*

- Shapes – Referencia o conjunto de objeto *ShapeObj* da janela
- Lines – Referencia o conjunto de linhas que interligam os objetos *ShapeObj*
- SelectedShapes – Referencia a lista de objetos *ShapeObj* que estão selecionados

Métodos de *DragDropWindow*

- CreateShape – Cria um objeto *ShapeObj* sem incluí-lo na janela
- CreateLine – Cria um objeto *LineObj* que interliga dois *ShapeObj*
- AddShape – Adiciona um objeto *ShapeObj* à janela
- AddLine – Adiciona um objeto *LineObj* à janela e liga dois *ShapeObj*
- RemoveShape – Remove um *ShapeObj* da janela

- RemoveLine – Remove um *LineObj* da janela
- SelectAll – Seleciona todos os objetos *ShapeObj*
- UnselectAll – Torna todos os objetos *ShapeObj* não selecionados

Em termos gráficos, dizemos então que um elementos E-DART é composto por uma representação gráfica (objeto *ShapeObj*) que pertence a uma janela gráfica (objeto *DragDropWindow*). Assim, temos o seguinte diagrama para os elementos E-DART que possuem representação gráfica.

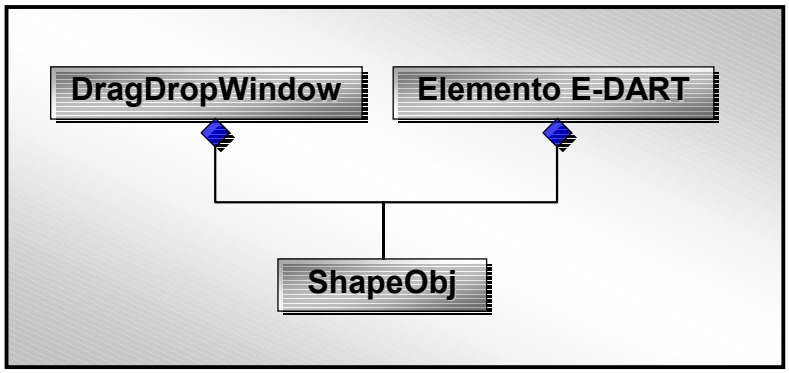


FIGURA B.2 - Relacionamento entre elementos E-DART e gráficos

Os objetos que modelam os elementos E-DART que possuem representação gráfica, além de referenciarem objetos gráficos, devem possuir métodos e propriedades padrão para que modelem de forma correta os elementos E-DART e possam ser tratados de forma uniforme pela ferramenta. Assim, definiu-se uma classe de objetos base para elementos E-DART chamado de *EDartBaseClass*. Tal classe modela os seguintes métodos e propriedades comuns aos objetos E-DART com representação gráfica.

Propriedades de *EDartBaseClass*

- UIDName – Identificador único do objeto
- Shape – Referência ao objeto gráficos *ShapeObj* correspondente
- Before – Referência ao objeto gráfico E-DART anterior
- After – Referência ao objeto gráfico E-DART posterior
- Process – Referência ao processo que contêm o elementos E-DART
- ReadyToDelete – Indica se o objeto pode ser excluído por uma ação do usuário

Métodos de *EDartBaseClass*

- PopupMenu – Exibe o menu sensível ao contexto correspondente
- Delete – Remove o objeto da estrutura interna de especificação
- AddBefore – Adiciona o objeto gráfico E-DART anterior
- RemoveBefore – Remove o objeto gráfico E-DART anterior
- AddAfter – Adiciona o objeto gráfico E-DART posterior
- RemoveAfter – Remove o objeto gráfico E-DART posterior
- ColorELotos – Adiciona código E-LOTOS na janela do E-LOTOS Just in Time
- SelectELotosText – Seleciona o código E-LOTOS na janela do E-LOTOS Just in Time
- ShowInPropBag – Mostra as propriedades do objeto na tabela de propriedades do ambiente
- Load – Lê a porção que descreve o objeto do arquivo de entrada
- Save – Grava a porção que descreve o objeto no arquivo de saída
- UpdateLinks – Atualiza as referências do objeto após sua leitura do arquivo de entrada

Para os objetos que modelam elementos E-DART e que possuem propriedades que podem ser alteradas pelo usuário através da tabela de propriedades do ambiente tem-se a seguinte característica: tais objetos devem implementar a interface da tabela de propriedades definida pela classe *Props*. Assim, como quase todos os elementos E-DART que possuem representação gráfica também possuem uma tabela de propriedades associada temos o seguinte gráfico.

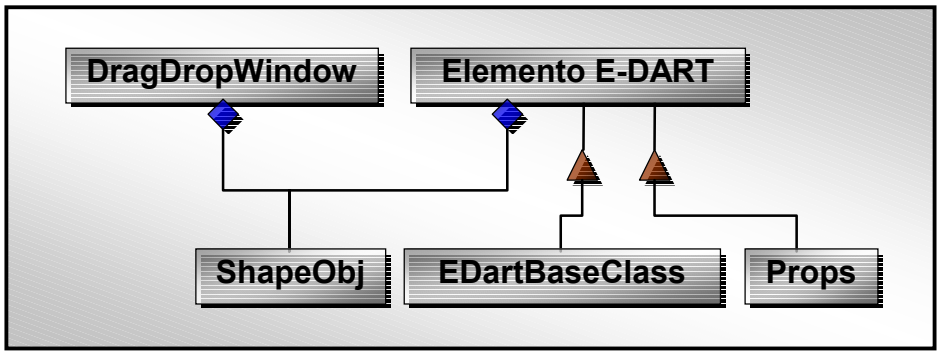


FIGURA B.3 - Elementos E-DART gráficos e tabela de propriedades

A classe *Props* possui os métodos a seguir.

Métodos de *Props*

- PropBagWrite – Escreve as entradas na tabela de propriedades do ambiente
- CellUpdate – Chamada toda vez que um valor de propriedade foi alterado
- BeforeCellChange – Procede com as ações que devem ser tomadas antes que uma entrada da tabela de propriedades do ambiente seja acessada

O restante dos objetos que modelam uma especificação são construídos conforme sua função na estrutura interna e não seguem nenhum padrão de construção. A seguir apresentamos o diagrama que cobre a maior parte das classes definidas na ferramenta para representar uma especificação internamente.

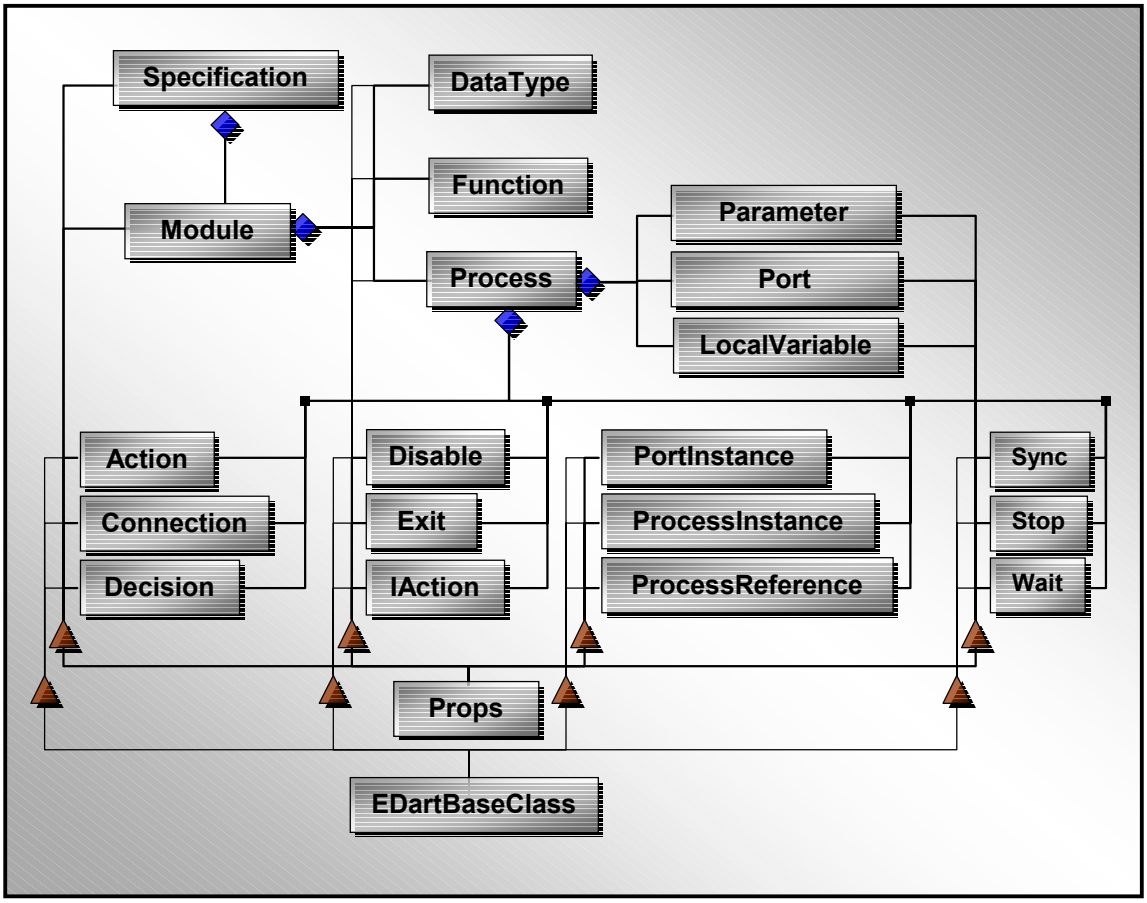


FIGURA B.4 - Classes do Editor E-DART

Especificação do Editor E-DART

Esta seção apresenta a especificação do Editor E-DART através da própria sintaxe gráfica E-DART. É também apresentado o código E-LOTOS correspondente aos principais processos da especificação. Cabe lembrar ainda que o foco desta especificação está voltado ao comportamento dinâmico da ferramenta e não aos dados internos à mesma.

Iniciamos a especificação apresentando o nível mais alto de abstração do sistema. Neste nível definimos os principais processos que representam os recursos do editor e os pontos de sincronização necessários para que exista a comunicação entre estes processos. A figura abaixo apresenta então a especificação deste níveis mais abstrato da ferramenta.

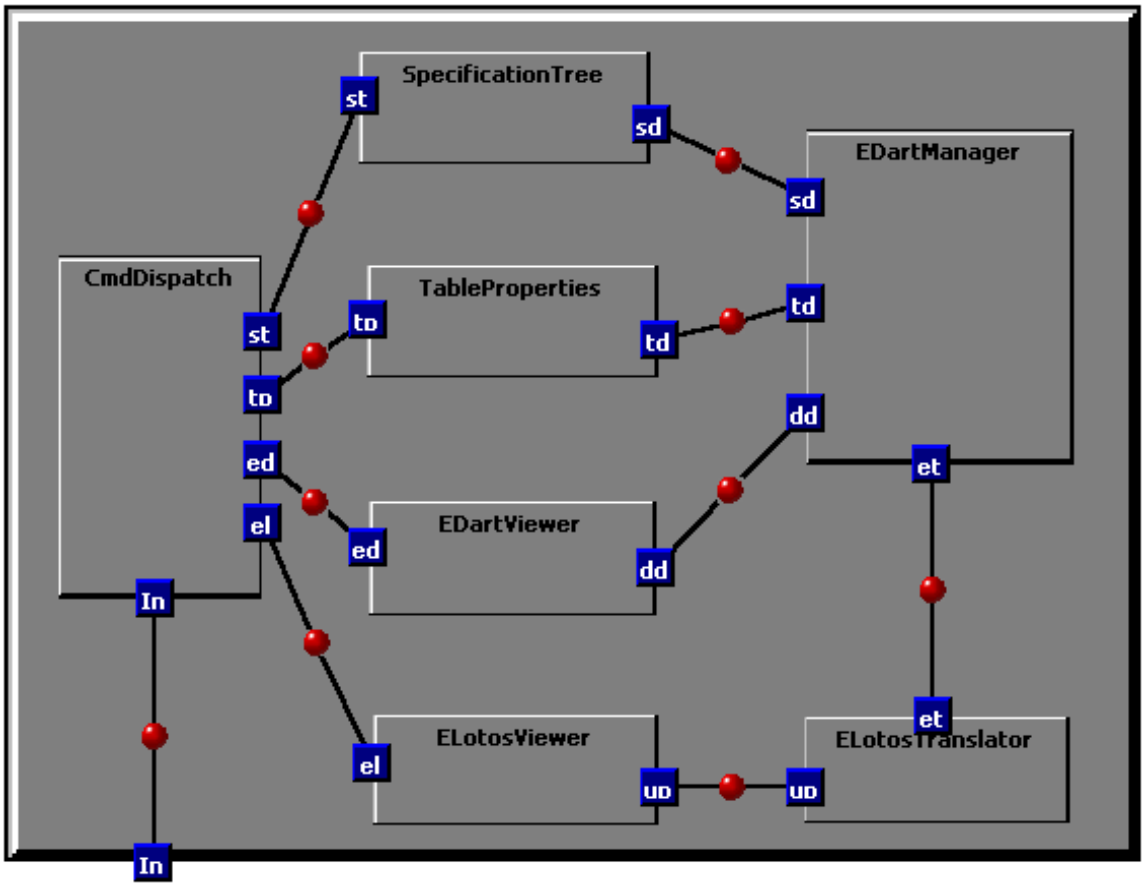


FIGURA B.5 - Nível mais abstrato da especificação do Editor E-DART

Nesta figura podemos observar como ocorre o fluxo de interação do usuário com a ferramenta. As interações do usuário ocorrem pela porta *In*. Uma sincronização em *In* ocorre com o processo *CmdDispatch* que tem por finalidade filtrar as interações do usuário e proceder com a comunicação aos elementos de interface corretos. Os dados que trafegados em *In* são do tipo *UsrCmd,* que descreve a ação que o usuário está tomando. Esta pode ser a ativação de um botão da barra de ferramentas, o clique do mouse em alguma região da tela, ou a escolha de uma opção de um item de menu.

O processo *CmdDispatch* é encarregado de encaminhar a ação do usuário para o elemento de interface correto. Tais elementos são representados pelo processos *SpecificationTree, TableProperties, EDartViewer* e *ELotosViewer.* Cada um deles modela as funcionalidades de um controle da ferramenta. A figura a seguir apresenta a especificação do processo *CmdDispatch*.

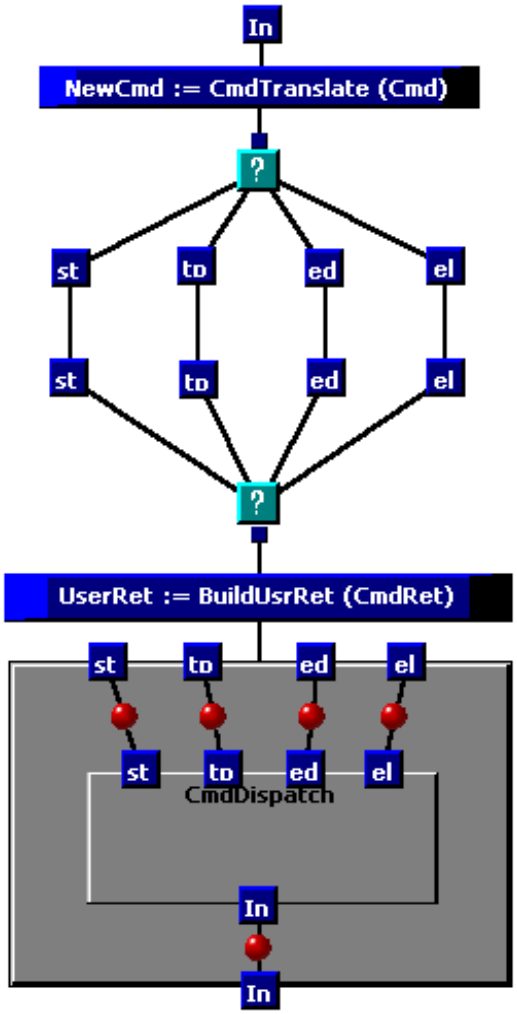


FIGURA B.6 - Processo *CmdDispatch*

A primeira sincronização do processo é feita na porta *In* que ocorre diante de uma ação do usuário. A variável *Cmd* recebe o comando ativado e *NewCmd* recebe o retorna da função *CmdTranslate,* que constrói dados estruturados para o comando requisitado. O retorno de *CmdTranslate* e *NewCmd* é do tipo *ResourceCmd*.

A segunda sincronização realizado pelo processo envia *NewCmd* para o processo cujo recurso de interface responderá ao comando. A seleção do correto processo é feita pelo operador de escolha, sendo que as sincronizações deste operador ocorrem apenas se *NewCmd.it,* um campo de *NewCmd*, identificar corretamente o destino da mensagem. Isso é possível porque o operador de sincronização pode restringir uma troca de mensagem baseado na avaliação de expressões. Assim, no

nosso caso, para a sincronização na porta *st*, que é o canal para a árvore de especificação, é definida como sendo:

```
st (!NewCmd) [NewCmd.it = "SpecificationTree"]
```

A próxima sincronização ocorre para proceder com o recebimento de informações de retorno da execução de uma operação. Este retorno também é armazenado em *NewCmd.* Já *UserRet* recebe o retorno de *BuildUserRet* que forma a ação de retorno para o usuário. Esta ação pode ser, por exemplo, a abertura de um menu sensível ao contexto. Depois disto, temos uma nova instanciação de *CmdDispatch,* que garante o tratamento das próximas interações do usuário.

Como dito anteriormente, os processos *SpecificationTree, TableProperties, EDartViewer* e *ELotosViewer* são responsáveis pelo controle dos elementos da interface da ferramenta. Estes processos recebem os comandos filtrados por *CmdDispatch* e retornam informações ao mesmo. Por outro lado, uma alteração nos dados dos elementos de interface muitas vezes deve refletir na alteração de uma estrutura interna guardada pela ferramenta. O processo *EDartManager* é aquele responsável por manter os dados internos relativos à especificação que o usuário está criando. *EDartManager* controla toda requisição de alteração nestas estruturas pelos outros processos da ferramenta. Logo, *SpecificationTree, TableProperties, EDartViewer* e *ELotosViewer* devem interagir com *EDartManager* toda vez que a mudança dos dados de um item de interface corresponder também a alteração dos dados internos.

Temos também o processo *ELotosTranslator* que é responsável por gerar código E-LOTOS que será apresentado pelas funcionalidades *DiagramTipsText, E-LOTOS Just in Time* e *Salvamento E-LOTOS* da ferramenta. Este processo comunica-se com *EDartManager* para garantir que toda alteração nas estruturas internas da ferramenta não implique na existência de código E-LOTOS inconsistente. Para isso, *ELotosTranslator* é ativado toda vez que uma modificação deve resultar em um novo código E-LOTOS correspondente. *ELotosTranslator,* por sua vez, tem acesso ao processo *ELotosViewer* para que as alterações do código E-LOTOS possam ser percebidas pelo usuário da ferramenta.

Assim, de forma geral, *SpecificationTree, TableProperties, EDartViewer* e *ELotosViewer,* comunicam-se por um lado com *CmdDispatch* e por outro com *EDartManager* ou *ELotosTranslator.* A figura a seguir apresenta a especificação do processo *ELotosViewer,* que é semelhante funcionalmente aos outros processos *SpecificationTree, TableProperties, EDartViewer.*

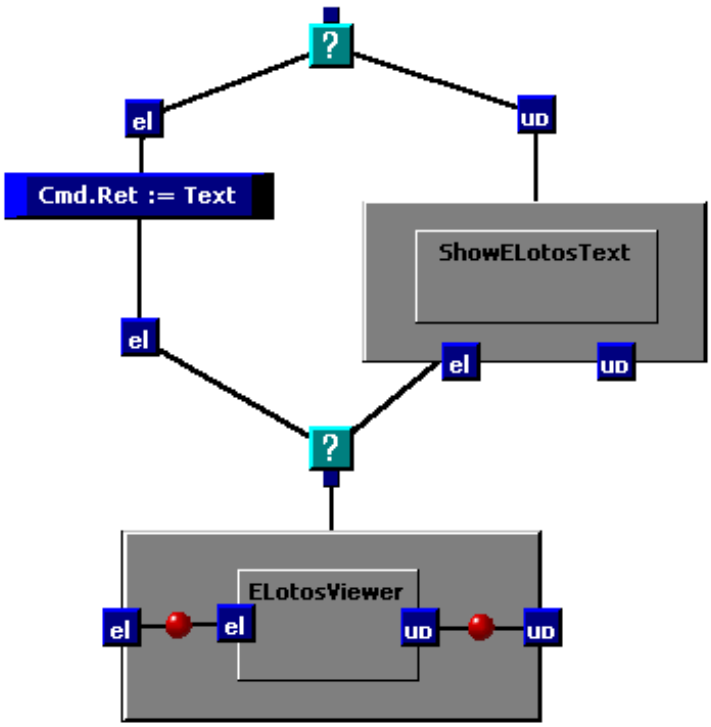


FIGURA B.7 - O processo *ELotosViewer*

Neste processo, a primeira ação é uma sincronização em *el* ou em *up*. A sincronização em *el* indica a requisição de execução de um comando do item de interface. Como nos códigos E-LOTOS apresentados pode-se apenas fazer a operação de *Copy*, a sincronização em *el* é definida como sendo:

```
el (?Cmd) [Cmd.de = "Copy"]
```

Assim, a variável *Cmd* só receberá uma informação se esta corresponder a uma ação de *Copy*. A variável *Text*, por sua vez, armazena o código E-LOTOS apresentado na interface da ferramenta. Assim, *Cmd.Ret* recebe este texto que é então repassado, através de uma segunda sincronização em *el*.

As sincronização pela porta *up* indicam que o processo *ElotosTranslator* gerou novo código E-LOTOS e este deve ser apresentado ao usuário. A sincronização em *up* é definida como:

```
up (?Text)
```

Assim, *Text* receberá o novo código E-LOTOS gerado e o processo *ShowELotosText* se encarregará de exibi-lo ao usuário.

Por fim, um novo processo *ELotosViewer* é instanciado recursivamente para permitir novas sincronizações em *el* e *up*. Passamos agora a apresentar a especificação do processo *EDartManager* representado pela figura a seguir.

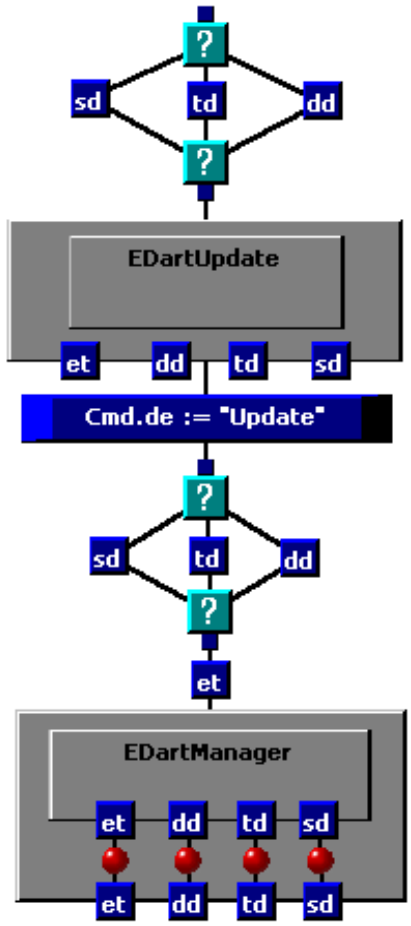


FIGURA B.8 - O processo *EDartManager*

A primeira ação em *EDartManager* é a sincronização com um dos processos *SpecificationTree, TableProperties* ou *EDartViewer*. Isto ocorre porque estes são os processos capazes de requisitar uma alteração nas estruturas internas da ferramenta. Após esta sincronização o processo *EDartUpdate* é instanciado para promover a alteração requisitada. Independente da alteração ser ou não executada, *EDartManager* sinaliza ao processo requisitor que este deve atualizar a apresentação de suas informações, refletindo o novo estado interno dos dados. Após isso, ocorre uma sincronização em *et* que indica ao processo *ELotosTranslator* que um novo código E-LOTOS deve ser gerado. Por fim, como aconteceu anteriormente, um novo processo é instanciado recursivamente para que novas sincronizações possam ser realizadas.

Apresentamos agora o código E-LOTOS, gerado pelo próprio Editor E-DART, para os gráficos mostrados anteriormente.

- **Nível mais abstrato da especificação do Editor E-DART**

```
specification E-DART Editor

      import
            UserInterface, GUIResources, E-LOTOS, E-DART is

      gates
            In: UserCmd

      behaviour
            hide st, tp, ed, el, sd, td, dd, up, et in
                  CmdDispatch [In, sp, tp, ed, el] ||| (
                        EDartManager [sd, td, dd, up] ||| (
                              SpecificationTree [st, sd] ||
                              TableProperties [tp, td]   ||
                              EDartViewer [ed, dd]       ||
                              (ELotosViewer [el, up] |[up]|
                                    ELotosTranslator [et, up]
                              )
                        )
                  )
            endhide
endspec
```

- **O processo *CmdDispatch***

```
process CmdDispatch [In: UserCmd, st: ResourceCmd, tp: ResourceCmd,
                  el: ResourceCmd, ed: ResourceCmd] : exit (none) is
      local
      var
            Cmd: UserCmd, NewCmd: ResourceCmd, CmdRet: ResourceCmd,
            UserRet: UserCmd
      in
            In (?Cmd);
            NewCmd := CmdTranslate (Cmd);
            (
                  st (!NewCmd) [NewCmd.it = "SpecificationTree"];
                  st (?CmdRet)
```

```
            ) [] (
                  tp (!NewCmd) [NewCmd.it = "TableProperties"];
                  tp (?CmdRet)
            ) [] (
                  ed (!NewCmd) [NewCmd.it = "EdartViewer"];
                  ed (?CmdRet)
            ) [] (
                  el (!NewCmd) [NewCmd = "ElotosViewer"];
                  el (?CmdRect)
            );
            UserRet := BuildUsrRet (CmdRet);
            CmdDispatch[In, st, tp, el, ed]
      endloc
endproc
```

- **O processo *ELotosViewer***

```
process ELotosViewer [el:ResourceCmd, up:ELotosText]:exit (none) is
      local
      var
            Text: ELotosText, Cmd: ResourceCmd
      in
            (
                  up (?Text);
                  ShowELotosText(Text)
            ) [] (
                  el (?Cmd) [Cmd.de = "Copy"];
                  Cmd.Ret := Text;
                  el (!Cmd)
            );
            ELotosViewer[el, up]
      endloc
endproc
```

- **O processo *EDartManager***

```
process EDartManager [sd: EDartCmd, td: EDartCmd,
            dd: EDartCmd, et: none] : exit (none) is
      local
```

```
        var
              Cmd: EDartCmd
        in
              (
                    sd (?Cmd)
              ) [] (
                    td (?Cmd)
              ) [] (
                    dd (?Cmd)
              );
              EDartUpdate(Cmd);
              Cmd.de := "Update";
              (
                    sd (!Cmd) [Cmd.it = "SpecificationTree"]
              ) [] (
                    td (!Cmd) [Cmd.it = "TableProperties"]
              ) [] (
                    dd (!Cmd) [Cmd.it = "EDartViewer"]
              );
              et;
              EDartManager[sd, td, dd, et]
        endloc
endproc
```

Bibliografia

[ALL 97] ALLENDE, Jesús S. **Graphical Designer for LOTOS**. Madrid: Dept. of Telematic Systems Engineering, Technical University of Madrid, 1994. Disponível por WWW em http://www.dit.upm.es/~jallende (maio 1997).

[AZC 89] AZCORRA, A.; QUEMADA, J.; FRUTOS, D. A Timed Calculus for LOTOS. In: FORTE, 1989, Vancouver, Canada. **Proceedings...** [S.l.:s.n.], 1989.

[BOL 87] BOLOGNESI, Tommaso; BRINKSMA, Ed. Introduction to the ISO specification language LOTOS. **Computer Networks and ISDN Systems**, Amsterdam, v.14, n.1, p.25-29, Jan. 1987.

[BOL 89] BOLOGNESI, Tommaso; BRINKSMA, Ed. Introduction to the ISO specification language LOTOS. In: VISSERS, Chris A.; VAN EIJK, Peter; DIAZ, Michel (Eds.). **The Formal Description Technique LOTOS.** Amsterdam: North-Holland, 1989. p. 303-326.

[BOL 93] BOLOGNESI, T.; LUCIDI, F.; TRIGILA, S. Converging towards a Timed-LOTOS Standard. **Computer Standards and Interfaces**, Amsterdam, p.87-118, 1993.

[BUD 87] BUDKOWSKI, S.; DEMBINSKI, P. An introduction to ESTELLE: A specification language for distributed systems. **Computer Networks and ISDN Systems**, Amsterdam, v.14, n.1, p.3-24, Jan. 1987.

[DAT 97] DATA DYNAMICS. **ActiveBar versão 1.0.2.9**. Disponível por WWW em http://www.datadynamics.com, 1997.

[EDA 98] E-DART Editor Home-Page. Diponível por WWW em http://www. inf.ufrgs.br/redes/edart e por E-mail em {granvile, trarbach}@ inf.ufrgs.br, 1998.

[EHR 85] EHRIG, H.; MAHR, B. **Fundamentals of Algebraic Specification**. Berlin: Springer – Verlag, 1985. v.1.

[GLO 89] ISO/IEC JTC1/SC21. **G-LOTOS: a Graphical Syntax for LOTOS**, N3253. 1989.

[GRA 95] GRANVILLE, Lisandro Zambenedetti. **Editor gráfico dos níveis estrutural abstrato e estrutural detalhado do SDL OO:** trabalho de diplomação. Porto Alegre: UFRGS, 1995.

[GRA 97A] GRANVILLE, Lisandro Zambenedetti; GASPARY, Luciano Paschoal; ALMEIDA, Maria Janilce. OST – An Object-Oriented Computer Networks Sytem Specification Tool. In: EUROPEAN CONFERENCE ON OBJECT ORIENTED PROGRAMMING,

ECOOP, 11., 1997, Jyväskyla, FI. **Proceedings...** Berlin: Springer-Verlag, 1997.

[GRA 97B] GRANVILLE, Lisandro Zambenedetti; GASPARY, Luciano Paschoal; ALMEIDA, Maria Janilce. OST – An Object-Oriented Computer Networks System Specification Tool. In: INTELLIGENT NETWORKS AND INTELLIGENCE IN NETWORK, 2IN97, 2., 1997, Paris, France. **Proceedings...** [S.l.: s.n.], 1997.

[GRA 98A] GRANVILLE, Lisandro Zambenedetti; ALMEIDA, Maria Janilce. E-DART: A Specification Environment to the E-LOTOS Formal Technique. In: JOINT WORKSHOP ON PARALLEL AND DISTRIBUTED REAL-SYSTEMS, WPDRTS, 1998, Orlando, U.S.A. **Proceedings...** [S.l.:s.n.], 1998.

[GRA 98B] GRANVILLE, Lisandro Zambenedetti; ALMEIDA, Maria Janilce. Definiçao e implementaçao do E-DART: Um Ambiente de Especificaçoes E-LOTOS. Em: WORKSHOP IBEROAMERICANO DE ENGENHARIA DE REQUISITOS E AMBIENTES DE SOFTWARE, IDEAS, 1998, Torres, RS. **Anais...** Porto Alegre, 1998. V.2, 236-248.

[GRA 98C] GRANVILLE, Lisandro Z.; ALMEIDA, Maria Janilce B. Graphical Support for the E-LOTOS Formal Technique in E-DART. In: THE OBJECT TECHNOLOGY CONFERENCE, OT, 1998, Oxford University, England. **Proceedings...** [S.l.:s.n.], 1998.

[GRA 98D] GRANVILLE, Lisandro Z.; ALMEIDA, Maria Janilce B. E-DART: A Specification Environment for the E-LOTOS Formal Technique. In: EUROPEAN CONCURRENT ENGINNERING CONFERECE, ECEC, 5., 1998, Erlangen-Nurember, Germany. **Proceedings...** Delft, The Netherlands: Society for Computer Simulation International, 1998.

[GRA 98E] GRANVILLE, Lisandro Z.; ALMEIDA, Maria Janilce B. E-DART: Um Ambiente para Especificação de Sistemas E-LOTOS. Em: 16º SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 1998, Rio de Janeiro, RJ. **Anais...** Rio de Janeiro: Instituto de Computação, Universidade Federal Fluminense, 1998.

[GRA 98F] GRANVILLE, Lisandro Z.; ALMEIDA, Maria Janilce B. E-Dart: A Specification Environment to the E-Lotos Formal Technique. In THE 22ND ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATION CONFERENCE, COMPSAC, 22., 1998, Viena, Áustria. **Proceedings...** [S.l.:s.n.], 1998.

[HEN 91] HENNESSY, M.; REGAN, T. A process algebra for timed systems. **Information and Computation**, Orlando, FL, 1991.

[HOA 85] HOARE, C.A.R. **Communicating Sequential Processes**. Englewood Cliffs: Prentice-Hall, 1985.

[ISO 97] ISO/IEC JTC1/SC21/WG7. **Working draft on enhancements to LOTOS**. Project WI 1.21.20.2.3, Jan. 1997.

[MIL 80] MILNER, Robin. **A Calculus of Communicating Systems**. NewYork: Springer-Verlag, 1980. (Lecture Notes in Computer Science, v.92).

[QUE 87] QUEMADA, J.; FERNANDES, A. Introduction of Quantitative Relative Time into LOTOS. In: IFIP WORKSHOP ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 7., 1987, Zurich. **Proceedings...** [S.l.:s.n.], 1987.

[SAR 87] SARACCO, R.; TILANUS, P.A.J. CCITT SDL: Overview of the language and its applications. **Computer Networks and ISDN Systems**, Amsterdam, v.13, n.2, p.65-74, 1987.

[REG 91] REGAN, T. **Process Algebra for Timed Systems**. University of Sussex, 1991. PhD Thesis.

[REG 93] REGAN, T. Multimedia in Temporal LOTOS: a Lip-Synchronization Algorithm. In: IFIP WORKSHOP ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 3., 1993. **Proceeding...** [S.l.:s.n.], 1993.

[SHE 96] SHERICAN SOFTWARE SYSTEMS, INC. **SSDBGrid Data Widgets 2.0**. Disponível por E-mail em sales@shersoft.com, 1996.

[SHE 97] SHERIDAM SOFTWARE SYSTEMS, INC. **SSSplitter Active Threed**. Disponível por E-mail em sales@shersoft.com, 1997.