Comunicado 55
Técnico
Dezembro, 2003
Campinas, SP

ISSN 1677-8464

Uso da Linguagem de Especificação SDL como Alternativa ao Diagrama de Estados Proposto pela Linguagem UML

Carla Geovana do Nascimento Macário[1]

O processo de desenvolvimento de software de qualidade constitui uma tarefa difícil e diversas são as metodologias de desenvolvimento propostas que buscam sua sistematização de maneira a torná-lo mais simples e organizado. Dentre elas, a orientação a objetos é atualmente uma das mais difundidas, centrando-se nos objetos que irão compor o sistema. Um objeto é definido como um conceito ou uma abstração de alguma coisa, permitindo um mapeamento direto dos elementos que compõem o sistema no mundo real com os componentes da aplicação e oferecendo uma base real para a sua implementação nos computadores. Segundo Booch (1991), a definição de um sistema em termos de objetos, levando-se em conta conceitos como encapsulamento, abstração, modularidade e herança torna a atividade futura de manutenção menos complexa, além de permitir ganhos de produtividade através do reuso de software.

Uma metodologia de desenvolvimento orientada a objetos enfatiza a identificação e a organização dos conceitos, ou entidades, do sistema em questão no lugar da sua representação final em linguagem de programação (Rumbaugh et al., 1991), além de permitir a descrição do sistema em diferentes níveis de abstração, iniciando pelo sistema como um todo e chegando até as funções em si. Uma vantagem desta abordagem é que, por permitir uma representação direta do sistema no mundo real, o seu entendimento pelo usuário e também por toda a equipe de desenvolvimento torna-se mais fácil. Além disso, quando necessárias, as alterações nos processos geralmente ficam restritas apenas a alguns objetos, tornando a atividade de manutenção menos complexa.

Nos anos 80 e 90, o uso de metodologias orientadas a objetos cresceu bastante, surgindo diferentes abordagens como as apresentadas por Meyer (1988), Booch (1991) e Rumbaugh et al. (1991).

A UML - Unified Modeling Language (Object Management Group, 2003), é uma convergência das principais metodologias que surgiram nesta época, combinando os aspectos importantes de cada uma delas. É definida como uma linguagem de modelagem, e não uma metodologia, para visualizar, especificar, construir e documentar os artefatos do sistema (Heap & Richards, 2000). O mais importante é que, diferente de suas antecessoras, permite uma padronização de notação gráfica para os diagramas essenciais de análise e de projeto e tem como principal vantagem ser mais completa, apresentando notações menos ambíguas que as anteriores, indo ao encontro do exposto pelo The International Engineering Consortium (2003), que descreve a capacidade de produzir especificação e projeto rigorosos, com um conjunto de conceitos bem definidos, técnicas que produzam especificações não ambíguas, claras, precisas e concisas e que possam ter sua consistência avaliada como uma das chaves do desenvolvimento de um sistema.

Entretanto, a quantidade de diagramas que esta linguagem prevê é grande e, principalmente com relação à fase de projeto, é necessário o uso de mais de um tipo de diagrama para se ter uma especificação completa do comportamento do sistema.

Este trabalho apresenta a linguagem de especificação SDL -

[1] M.Sc. em Engenharia Elétrica e de Computação, Pesquisadora da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo - 13083-970 - Campinas, SP. (e-mail: carla@cnptia.embrapa.br)

Specification and Description Language - como alternativa ao diagrama de estados proposto pela UML para a fase de projeto. Serão abordadas as duas técnicas e os diagramas gerados por cada uma delas para um mesmo caso de uso. Ao final, estabelece um conjunto de critérios que permite a sua avaliação, discutindo-se os resultados obtidos.

Linguagem UML

Em uma metodologia orientada a objetos, diversos diagramas e modelos são produzidos durante a construção do software, sendo usados durante todos os estágios do processo de desenvolvimento de forma evolutiva, provendo uma descrição completa do sistema. Dentre esses modelos o de classes serve de base para os demais, descrevendo as classes dos objetos, seus relacionamentos, seus atributos e suas operações. Uma *classe* descreve um grupo de objetos com propriedades semelhantes (atributos), com o mesmo comportamento (operações) e relacionamento com outros objetos e com a mesma semântica. As *classes dinâmicas* são aquelas que exercem alguma ação sobre outra classe, e as *passivas* são aquelas que apenas sofrem uma ação. Mais informações sobre classes e objetos podem ser encontrados em Rumbaugh et al. (1991).

A linguagem UML estabelece passos a serem seguidos para a obtenção de modelos e de diagramas a serem gerados em 3 fases distintas do desenvolvimento de software - análise, projeto e implementação.

A *análise* é a fase de concepção do sistema, buscando-se o entendimento e a especificação do que ele deve fazer. Os principais passos desta fase são a identificação dos atores e dos limites do sistema, o desenvolvimento dos casos de uso e dos cenários e a construção do modelo de objetos do domínio. Tudo isso deve ser feito visando capturar as classes de negócio, seus relacionamentos e seu comportamento, incluindo regras e vocabulário do negócio. Talvez seja uma das fases que exija maior esforço, já que pretende-se ter como resultado a definição clara daquilo que o cliente deseja. Os produtos gerados nesta fase são o *modelo de casos de uso* e o *modelo de classes do domínio*.

Já na fase de *projeto* descreve-se como as classes identificadas na fase anterior irão realizar seus objetivos, funcionando como uma ponte entre a fase de análise e a de implementação. Neste momento devem ser consideradas as restrições de implementação e também os conceitos de objetos como reuso, herança, abstração e encapsulamento. Aqui são definidas as interfaces entre as classes e suas interações e também a maneira como estas devem executar suas ações para atingir os objetivos propostos na fase de análise. Os passos-chave do projeto são: estudar as restrições de implementação e definir a arquitetura técnica; identificar as classes de projeto e atribuir suas responsabilidades; criar diagramas de interação e validar o modelo de classes; completar e refinar o modelo de classes e produzir a especificação completa das classes. Ao final, tem-se como produtos especificações e diagramas detalhados das classes, diagramas de seqüência e de estado, definição das interfaces e dos subsistemas e descrição da arquitetura do software.

Por fim a fase de *implementação* consiste na construção do sistema em si, sendo a implementação e teste dos modelos definidos. Os passos principais são a geração ou a definição das classes na linguagem escolhida, a implementação do modelo lógico, a construção dos testes unitários e a integração das classes e seu teste.

Fase de Projeto na UML: o Diagrama de Estados

Como citado anteriormente, no projeto pretende-se criar um modelo da interação dos objetos que satisfaçam aos requerimentos estabelecidos e que possa ser implementado. Parte desta atividade é feita através do diagrama de estados.

Durante a execução de um cenário, o objeto pode assumir diferentes estados. Cada *estado* representa o objeto em um determinado momento de sua execução e é descrito em termos do valor de seus atributos e de seu relacionamento com outros objetos. A ação que provoca uma mudança de um estado para outro é denominada *evento*, sendo que nem todas as ações provocam a mudança de estado. O diagrama de estados é uma ferramenta que apresenta todos os estados que uma classe poderia assumir durante a execução do sistema e permite a essa classe saber exatamente como reagir a um evento. Além disso, este diagrama ajuda a avaliar se todas as ações possíveis para uma classe foram identificadas, a descrever a vida de um objeto para uma dada classe, a descrever todas as possibilidades de resposta da classe aos eventos e identificar estados onde não existe uma ação associada para a mudança. Uma grande confusão é feita em relação ao seu escopo: cada diagrama de estados descreve uma única classe do sistema e nunca para o sistema com um todo, devendo ser feito para todas as classes que apresentem comportamento dinâmico.

A notação utilizada para a construção de diagramas de estados na UML é baseada em Harel (1987), com algumas variações para torná-la orientada a objetos, como pode ser visto na Fig. 1. O estado inicial indica a criação de uma instância da classe, o objeto, e o estado final a sua exclusão. Alguns objetos, entretanto, podem nunca serem excluídos em uma execução do sistema.

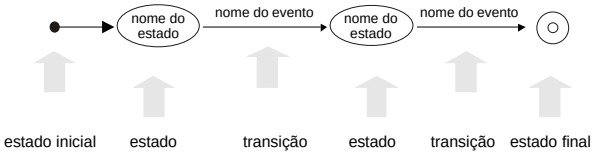

Fig. 1. Notação básica de diagrama de estados em UML. Fonte: Heap & Richards (2000).

Os passos para construção de um diagrama de estados, resumidamente, são: identificar os estados que fazem parte do fluxo básico do caso de uso; identificar as ações alternativas, que são aquelas definidas para tratar exceções; identificar as ações de transição, que são executadas na ocorrência de um evento; identificar ações de execução, que são executadas durante um estado; identificar ações de entrada do estado *(entry actions*), que são executadas

sempre que o objeto entra em um estado; identificar ações de saída de estado *(exit actions)*, que são executadas imediatamente antes do objeto sair de um estado; identificar condições de guarda (*guard conditions*), uma expressão booleana feita em termos das variáveis de estado do objeto e usada para indicar a ocorrência ou não de mudança de estado; identificar transições internas, que são executadas durante um estado e inibem a execução das ações de entrada e de saída; identificar transições internas, que também são executadas durante um estado (*do actions*), mas não inibem a execução das ações de entrada e de saída; e identificar as variáveis de estado, manipulando seu conteúdo quando necessário

A seguir é apresentado um exemplo do diagrama de estado para a classe *caixa* definida em um caso de uso *Saque em caixa-automático de um banco*. Maiores informações sobre casos de uso podem ser obtidas em Heap & Richards (2000) e em Object Management Group (2003).

A Fig. 2 apresenta a representação da classe *caixa* no fluxo básico deste caso de uso e corresponde ao primeiro passo da construção de um diagrama de estados. Nesta Fig. estão representados os estados que a classe pode assumir e os eventos que provocam a mudança de um estado para outro.

A Fig. 3 apresenta o diagrama final, após passar pelos demais passos da sua construção.

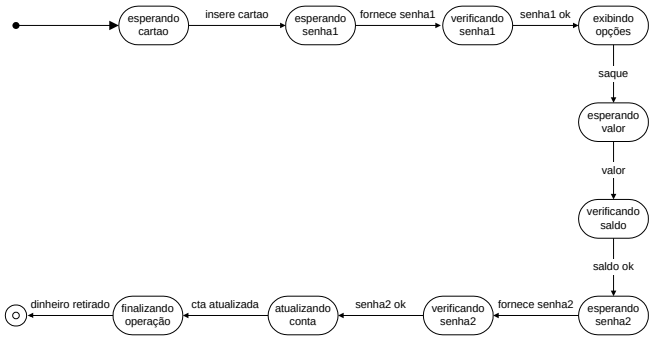

Fig. 2. Diagrama de estados classe caixa - fluxo básico.

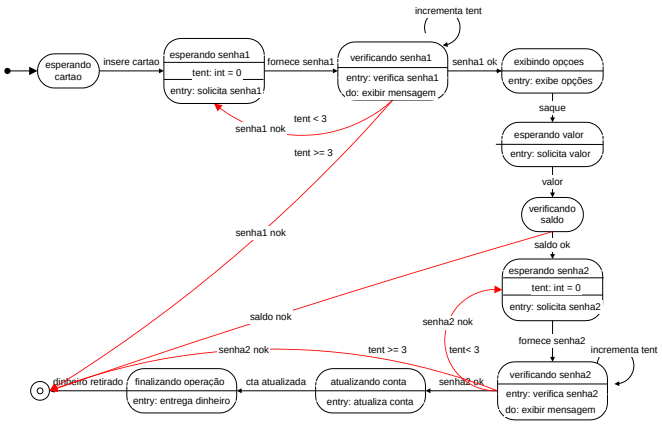

Fig. 3. Diagrama de estados classe caixa - completo.

Linguagem SDL

A Specification and Description Language (SDL) é uma linguagem formal e orientada a objetos definida pela ITU-T (1999), antiga CCITT, cujo desenvolvimento iniciou-se em 1972 para a especificação e a descrição de sistemas de telecomunicações. Atualmente é usada em sistemas complexos dirigidos por eventos de tempo real e interativos, e que envolvam atividades concorrentes. Sua primeira versão foi lançada em 1976, tendo sido seguida de novas versões lançadas a cada quatro anos (Belina et al., 1991). A versão lançada em 1992, denominada SDL-92, representou um marco na sua história por incorporar conceitos de objetos, como tipos, reuso e herança, ampliando a sua abrangência, com a tendência de ser utilizada na especificação dos mais diversos produtos de diferentes áreas, principalmente em sistemas de tempo real, interativos ou distribuídos (Faergemand & Olsen, 2003). Atualmente é uma linguagem de padrão internacional, sendo utilizada em empresas de diversos países e tendo com principais áreas de atuação: comunicação de satélites, aeronáutica, equipamentos médicos, sistemas de controle de estradas de ferro e protocolos de comunicação usados nos carros (The International Engineering Consortium, 2003).

A linguagem SDL serve tanto para representar o comportamento do sistema, como sua estrutura, usando diferentes níveis de abstração, do mais alto ao mais detalhado. Uma de suas características fortes é que, diferente da maioria das linguagens de especificação formal que apresentam apenas uma representação textual, a SDL possui também uma representação gráfica, denominada SDL-GR, baseada em símbolos, o que provê clareza e facilidade de uso, facilitando bastante o seu entendimento. Além disso, por ser formal, possui uma sintaxe e uma semântica bem definidas, possibilitando a simulação e a validação automática do sistema sendo especificado, mesmo nas fases iniciais do desenvolvimento, através de uma ferramenta CASE.

Uma especificação SDL provê diversas visões de um sistema, sendo elas:

Visão estrutural, através da:

- decomposição hierárquica do software em sistema, blocos e processos;
- hierarquia de tipos, através de herança e especialização (apenas da versão SDL-92 em diante).

A visão estrutural permite o particionamento do sistema em diferentes níveis hierárquicos e tem como principais objetivos: ocultar informação, movendo detalhes para níveis mais baixos, seguir as subdivisões naturais, criar módulos de tamanhos gerenciáveis, criar correspondência com o hardware e o software do mundo real, reusar especificações prontas. Nesta decomposição, SDL provê interfaces bem definidas entre as diversas entidades do sistema. A Fig. 4 ilustra a decomposição de um sistema na linguagem SDL.

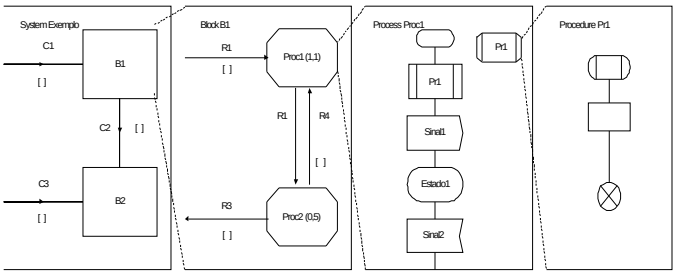

Fig. 4. Decomposição hierárquica de um sistema na linguagem SDL.

Visão de comunicação, através de:

- sinais assíncronos e seus parâmetros opcionais;
- chamadas remotas a procedimentos (apenas da versão SDL-92 em diante).

Visão comportamental:

- é descrita através dos processos.

Visão de dados através de:

- tipos abstratos de dados (sorts) definidos;
- tipos de dados ASN.1[2].

Dentre estas visões, a comportamental é a que apresenta-se como alternativa ao diagrama de estados proposto pela UML.

SDL - Visão Comportamental

A visão comportamental do sistema descrito em SDL é dada através dos processos. Um processo em SDL é uma máquina de estados finita e pode ser criado no início ou durante a execução do sistema e podem existir ao mesmo tempo uma ou mais instâncias do processo, tendo cada uma delas um identificador único. A comunicação entre os processos se dá através de sinais, que podem ou não utilizar parâmetros. A troca de sinais é sempre entre dois processos e nunca entre vários deles. A Fig. 5 ilustra a representação de um processo numa especificação SDL.

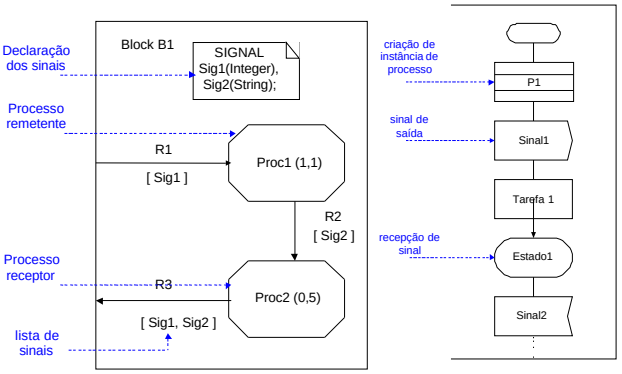

Fig. 5. Representação de um processo numa especificação SDL.

Para a descrição do comportamento de um processo é utilizado um conjunto de símbolos que compõem a linguagem gráfica SDL. Os principais deles são apresentados a seguir.

	Início da execução do processo		término da execução do processo
Estado1	representa o estado, cujo nome está escrito no símbolo; em especial o uso de **-** no seu nome indica o estado não se altera após a execução de alguma ação e o uso de ***** refere-se a todos os estados, estabelecendo ações comuns a todos eles; a mudança de estado se dá através da recepção de um sinal.	P1	criação de instância de um processo do mesmo bloco
Tarefa 1	tarefa, usada para definir uma ação; pode ser um 'texto' ou um comando	decisão	decisão; contém uma questão e um determinado número de respostas
Sinal2	sinal de entrada; vem sempre depois de um estado	Sinal1	sinal de saída, usado para enviar sinais e dados para outros processos
Pr1	chamada a procedimento		início da execução do procedimento
	término da execução do procedimento		comentário
cnt1	conector, usado para conectar partes do processo		

[2] ANS.1 - Abstract Syntax Notation One , linguagem definida pela ITU-T para a descrição de tipos de dados e de valores, muito utilizada na especificação de dados em protocolos e serviços de telecomunicações, especialmente os que envolvem os níveis de aplicação. Muitos dos padrões da área de telecomunicações são baseados nesta linguagem.

A Fig. 6 apresenta a especificação do comportamento da classe caixa usada anteriormente como exemplo do diagrama de estados (Fig. 3).

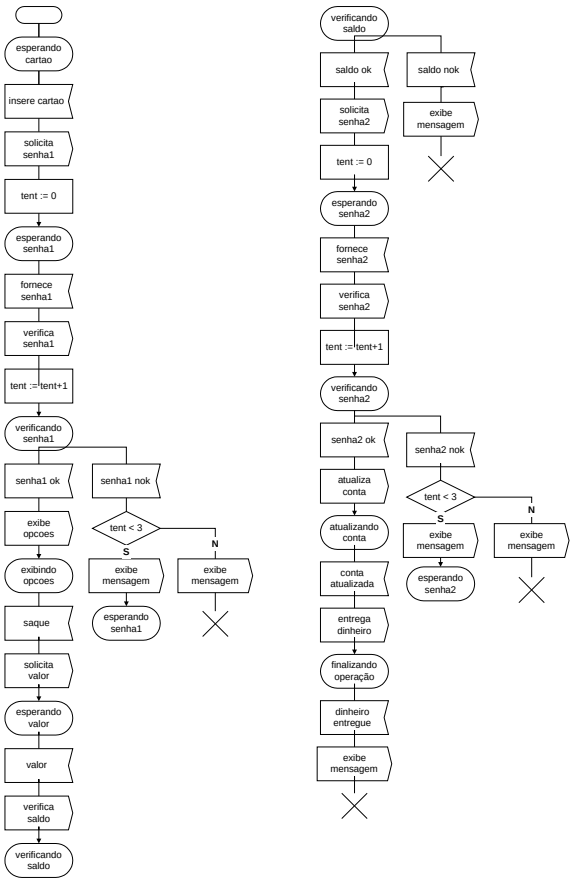

Fig. 6. Diagrama SDL da classe Caixa.

Comparação entre as Duas Técnicas Apresentadas

Foram definidos alguns critérios para que fosse possível comparar as duas técnicas. São eles: número de estados: números de estados definidos em cada diagrama, número de símbolos: número de símbolos gráficos utilizados na especificação da classe; complexidade da informação de cada símbolo: quantidade de informação que cada símbolo apresenta, podendo ser alta, média ou baixa; tratamento de decisões: indica se a técnica apresenta mecanismos para o tratamento adequado de decisões; tratamento de exceções: indica se a técnica apresenta mecanismos para o tratamento adequado de exceções, especificação das ações: indica se a técnica apresenta mecanismos para a especificação das ações relacionadas aos estados; fluxo de controle: indica se a técnica permite que se tenha uma idéia do fluxo de controle que o objeto pode apresentar; interação com outros objetos: indica se a técnica permite identificar todas as interações da classe com outras classes. Considerando-se estes critérios, têm-se os resultados apresentados na Tabela 1.

Tabela 1. Análise dos critérios apresentados.

Critério	*Diagrama de estados UML*	*SDL*
Número de estados	10	10
Número de símbolos	10	52
Complexidade da informação de cada símbolo	alta	baixa
Tratamento de decisões	presente	presente
Tratamento de exceções	presente	presente
Especificação das ações do estado	presente	presente
Fluxo de controle	ausente	presente
Interação com outros objetos	baixa	alta

É de se esperar que o número de estados obtidos com o uso de cada técnica seja o mesmo, já que se trata da especificação de comportamento para uma mesma classe. Com relação ao número de símbolos usados, em SDL tem-se um número bem maior que no diagrama da UML. Entretanto, este item deve ser avaliado em conjunto com a complexidade dos nós. Dado que ambos diagramas especificam a mesma coisa, e um deles utiliza um número bem menor de símbolos, a quantidade de informação que o símbolo do diagrama da UML carrega deve ser maior que a do diagrama SDL, tornando-o mais complexo e talvez, em alguns casos, dificultando a sua elaboração e o seu entendimento. O tratamento de decisões e de exceções está presente nos dois, mostrando que ambos atendem situações que fogem ao fluxo básico de uma classe. Mas não é claro como especificar uma ação associada a uma exceção, como por exemplo a exibição de mensagens de erro. Parece que o diagrama UML não é completo neste sentido. Em SDL isto é feito como a especificação de uma ação comum. Com relação ao fluxo de controle do objeto, isso não existe no diagrama de estados da UML apenas na linguagem SDL. Na verdade, para isso a metodologia UML prevê o uso do diagrama de atividades, uma variação de uma máquina de estados, na qual os estados representam a execução de ações ou de subatividades e as transições são disparadas após a sua execução completa (Object Management Group, 2003). Entretanto, este diagrama é construído para cada cenário do sistema, ou seja, representa uma execução do sistema. A linguagem SDL, por ser estruturada, provê as duas visões em um só diagrama, especificando todos os cenários possíveis. Por fim, com relação à interação com os demais objetos, no diagrama de estados da UML algumas das mensagens trocadas com outras classes não aparecem. Já o diagrama SDL inclui todas as mensagens possíveis de serem enviadas e recebidas pela classe.

Conclusões

A adoção de uma metodologia de desenvolvimento orientada a objetos facilita o desenvolvimento de um sistema de software por permitir o seu mapeamento direto com o mundo real e também pelas características que a orientação a objeto apresenta (abstração, encapsulamento, herança, reuso). A

resposta às diversas metodologias existentes na década de 90 e tem como principais vantagens ser mais completa e com notações menos ambíguas que suas antecessoras. Entretanto, a quantidade de diagramas que ele prevê é grande e principalmente com relação à fase de projeto, é necessário o uso de mais de um tipo de diagrama para se ter uma especificação completa do comportamento do sistema.

Este trabalho apresentou o uso da linguagem SDL como alternativa ao diagrama de estados da linguagem UML. SDL é uma linguagem formal e orientada a objetos para a especificação e a descrição de sistemas complexos dirigidos por eventos de tempo real e interativos, e que envolvam atividades concorrentes, especialmente na área de telecomunicações.

Para que fosse possível uma avaliação das duas técnicas, propôs-se um conjunto de critérios comparativos, focando a dificuldade e a completude de cada técnica. Neste sentido, o diagrama SDL mostrou-se mais completo e mais simples. Além disso, é capaz de reunir a informação de mais de um diagrama existente na linguagem UML. Assim, o seu uso permite, diminuir o número de diagramas a serem gerados na especificação do sistema, o que tem como efeito direto maior agilidade na fase de especificação e de evolução.

Referências Bibliográficas

BELINA, F.; HOGREFE, D.; SARMA, A. SDL with applications from protocol specification. Englewood Cliffs: Prentice Hall International, 1991. 275 p.

BOOCH, G. Object oriented design with applications. Redwood City: Benjamin Cummings, 1991. 578 p.

FAERGEMAND, O.; OLSEN, A. New features in SDL-92. Disponível em: < ftp://ftp.sdl-forum.org/pub/papers/SDL92AOOF.doc > . Acesso em: 25 jul. 2003.

HAREL, D. Statecharts: a visual formalism for complex systems. Science of Computer Programming, v. 8, p. 231-274, 1987.

HEAP, K.; RICHARDS, T. Object-oriented analysis and design using the Unified Modeling Language: student guide. Redwood Shores: Oracle Corporation, 2000. Paginação irregular. (Oracle University).

THE INTERNATIONAL ENGINEERING CONSORTIUM. Specification and Description Language (SDL). Disponível em: < Http://www.iec.org/online/tutorials/sdl/sdl_iec_1809.pdf > . Acesso em: 18 set. 2003.

ITU-T. Recommendation Z.100 - specification and description language (SDL). Geneve, 1999.

MEYER, B. Object-oriented software construction. New York: Prentice Hall, 1988. 534 p. OBJECT MANAGEMENT GROUP. Unified Modeling Language (UML), version 1.5. Needham, 2003. Paginação irregular.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. Object oriented modeling and design. New Jersey: Prentice Hall International, 1991. 500 p.

Comunicado Técnico, 55

Ministério da Agricultura, Pecuária e Abastecimento
Governo Federal

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Endereço: Caixa Postal 6041 - Barão Geraldo
13083-970 - Campinas, SP
Fone: (19) 3789-5743
Fax: (19) 3289-9594
e-mail: sac@cnptia.embrapa.com.br

1ª edição on-line - 2003