ORIENTAÇÃO PARA OBJETOS EM SMALLTALK-80 - UMA ABORDAGEM EFICAZ PARA A CONSTRUÇÃO DE SISTEMAS DE SOFTWARE.

Miguel Jonathan

Núcleo de Computação Eletrônica (UFRJ)
Departamento de Ciência da Computação (IM/UFRJ)
Caixa Postal 2324
20001 Rio de Janeiro - RJ

SUMARIO

Este artigo tem por objetivo apresentar alguns conceitos e características da programação orientada para objetos e, em especial, da linguagem Smalltalk-80, e analisar as perspectivas de sua utilização. As facilidades inerentes para re-utilização de software, construção de protótipos rápidos, e a conseqüente redução dos custos de desenvolvimento e manutenção, com aumento de confiabilidade, sugerem que essa abordagem pode contribuir significativamente para a solução da chamada "crise do software".

ABSTRACT

This paper presents some concepts and features of object-oriented programming and of the Smalltalk-80 language in particular, and analyzes the perspectives of its use. The facilities for software reusability and for building rapid prototypes, and the consequent reduction in development and maintenance costs, with increased reliability, suggest that this approach may contribute in a significant way for the solution of the so-called "software crisis".

1. Introdução.

1.1 A Crise de Software.

Nos últimos anos tem havido um interesse cresente por novas abordagens de programação que permitam superar a chamada "crise de software" - isto é, a dificuldade de construir sistemas de software complexos e confiáveis, com baixo custo de construção, manutenção e adaptação, e de explorar eficazmente as possibilidades de distribuição de processamento e paralelismo em larga escala abertas com os avanços no desenvolvimento de VLSI e das redes de comunicação de dados.

As linguagens algoritmicas procedurais convencionais (Fortran, Algol, Pascal, C e outras) baseiam-se em um paradigma de programação que segue de perto a arquitetura de uma máquina de von Neumann: a utilização do computador consiste na execução de programas; cada programa é uma sequencia de instruções executadas passo a passo, em um único processador; programas leem "dados" através de comandos de entrada, e produzem "resultados" através de comandos de saída. O computador de von Neumann foi idealizado para realizar computações matemáticas através de algoritmos (daí o seu nome) e para esse fim demonstrou ser extremamente eficiente. Mais tarde, porém, passou a ser usado também para representar e simular o comportamento de sistemas do mundo real tão diversos como sistemas de informação, controle de processos industriais, jogos educativos, animação gráfica e automação de escritórios. Para representá-los através das linguagens algoritmicas convencionais, é necessário um esforço grande de tradução do modelo sistêmico (interações de objetos) para um modelo algoritmico (programas e dados), com considerável perda de compreensibilidade. Mais ainda, os programas resultantes são de difícil verificação, e modificações e extensões nas especificações, embora pequenas, podem resultar em um custo elevado de implementação. Para amenizar essas dificuldades diversas técnicas tem sido desenvolvidas, como Análise Estruturada, Programação Modular, e utilização de

sub-sistemas especializados, como sistemas de gerencia de bancos de dados, editores de texto, planilhas eletrônicas, etc., que nem sempre são integrados á linguagem de programação, exigindo que sejam usados diversos ambientes de programação para uma mesma aplicação.

Este artigo visa apresentar alguns conceitos e caracteristicas básicas da programação orientada para objetos e, em especial, da linguagem Smalltalk-80 [Goldberg e Robson 83], e discutir as perspectivas de sua utilização como forma de contornar algumas das dificuldades mencionadas acima.

1.2 Um novo paradigma de programação.

Embora não seja uma idéia nova (a primeira implementação de Smalltalk data de 1972), a programação orientada para objetos só recentemente vem recebendo uma atenção mais ampla da comunidade de computação, como mostra a realização da primeira Conferência sobre Sistemas de Programação, Linguagens e Aplicações Orientadas para Objetos, realizada pela ACM em novembro de 1986 (OOPSLA'86) [Meyrowitz86]. Entre as dificuldades para sua utilização mais geral estão a falta de arquiteturas apropriadas e a força da cultura tradicional de programação que produz resistências a mudanças na forma de conceber e realizar sistemas de software (Robson (1981) observa que muitas pessoas sem experiência prévia de programação acham bastante natural a idéia de sistemas orientados para objetos, enquanto programadores experientes tendem inicialmente a estranhar essa concepcão).

Sistemas orientados para objetos são concebidos e implementados de forma mais próxima de como a mente humana percebe os sistemas da realidade: como uma interação entre objetos distintos, cada qual possuindo propriedades e comportamentos próprios, e onde cada objeto pode ter outros objetos como componentes de sua estrutura. Além disso, objetos com propriedades similares são reunidos em classes, com a possibilidade de se representar generalizações (super-classes) e especializações (sub-classes) de uma dada classe. Nesses

sistemas a distinção entre "dados" e "programas" desaparece e o ato de programar é dividido em duas partes: de um lado a definição das classes de objetos e suas propriedades e, de outro, a criação de instâncias de classes e sua manipulação através da troca de mensagens entre esses objetos, como será visto adiante na seção 2.

1.3 Um pouco de História.

Simula (Birtwistle et al.73), uma extensão da linguagem Algol 60, foi a primeira linguagem de larga divulgação a incorporar a concepção de uma hierarquia de classes de objetos, onde cada classe é um módulo englobando estrutura e comportamento comuns às suas instAncias. As idéias de Simula serviram de base para a teoria dos tipos abstratos de dados [Liskov e Ziller74] e para a linguagem Smalltalk, desnvolvida pela Xerox. Smalltalk recebeu também outra herança importante da parte de Alan Kay, um de seus idealizadores: o princípio de objetos ativos, sempre prontos a reagir a mensagens ativando um "comportamento" correspondente, e a idéia de estender o conceito de orientação para objetos de forma uniforme por todo o ambiente de software. Smalltalk é uma evolução da linguagem FLEX [Kay68, Kay69] e passou por várias versões até a atual, Smalltalk-80 [Goldberg e Robson83].

A partir dessa versão a linguagem extrapolou dos limites da Xerox e vem sendo implementada por diversos outros fabricantes em seus equipamentos (Apollo DOMAIN, SUN2, Tektronix 4406, VAX, etc. [Kresner83]) e em Universidades [Ungar e Paterson83, Samples, Ungar e Hilfinger86]. Versões reduzidas para o IBM-PC (Methods, da Digitalk) e PC-AT (Smalltalk-AT, para o processador 80386) também foram implementadas com sucesso.

2. Caracteristicas da linguagem Smalltalk-80.

2.1 Objetos e mensagens.

Smalltalk-80 é uma linguagem baseada no conceito de que toda computação é ativada através da troca de mensagens entre objetos. Cada objeto simula uma abstração que possui representação interna e uma memória privativa que caracteriza o seu estado, e pode "reagir" a mensagens que fazem parte do seu protocolo de comunicação com o exterior. Ao reconhecer uma mensagem, o objeto ativa o procedimento (ou método) específico que descreve o comportamento associado à mensagem.

Todo objeto é uma instância de alguma **classe**. A classe contém a descrição da representação interna e dos métodos comuns a todas as suas instâncias . Cada objeto em particular possui sua própria memória privativa que só ele pode acessar e modificar. Uma caracteristica única de Smaltalk-80 é o uso uniforme do paradigma de mensagens e objetos. Todos os conceitos são representados por objetos do sistema. Em particular, classes são também objetos (toda classe é uma instância da classe **Class**) e reagem a mensagens próprias

Para ilustrar, seja a classe **Ponto**. A classe contém uma descrição de suas instâncias que consiste no caso de 2 coordenadas retangulares **x** e **y**, cada qual um objeto da classe **Numero**. Uma instância de Ponto pode ser criada enviando-se a seguinte mensagem à classe Ponto (Para maior clareza, os comandos são escritos em Português):

```
Ponto novo
```

A classe Ponto reage a esta mensagem criando uma nova instância e retornando uma referência a ela para o objeto que enviou a mensagem.
Note que **novo** é uma mensagem compreendida pela classe Ponto, e não pelas instâncias. Instâncias de Ponto, por sua vez, reagem a mensagens como:

```
x: 100  y: 150
```

atribuindo-se as coordenadas contidas nos argumentos x e y.

A mensagem é identificada pelo seu **seletor**, que no caso é **x:y:**
Para criar uma nova instância de Ponto com essas coordenadas basta encadear as mensagens:

```
Ponto novo  x: 100  y:150
```

Seja agora a classe Circulo. Supondo que Ponto esteja implementada, Circulo pode descrever suas próprias instâncias através de duas variáveis: **centro**, que é uma instância de Ponto, e **raio**, que é um Número. A classe Circulo pode também reagir a uma mensagem **novo**, para produzir uma nova instância. Cada instância por sua vez reage a uma mensagem com seletor **centro:raio:** para se atribuir um centro e um raio. Como acima, é possivel encadear as mensagens:

```
Circulo novo  centro:(Ponto novo x:100 y:150)  raio:10
```

Mensagens e seus métodos associados podem ser acrescentados ou retirados do repertório de cada classe de forma dinâmica. De certa forma pode-se fazer a analogia de que uma vez criada a classe, cada novo método implementado equivale a "ensinar" aos objetos um novo "comportamento" ou "habilidade" como, por exemplo, ensinar os circulos a mudar de posição na tela ou a alterar o seu tamanho.

A interpretação de uma mensagem recebida depende exclusivamente da classe do objeto que a recebe. Uma mesma mensagem recebida por dois objetos de classes diferentes pode ser interpretada de forma diferente, desde que cada classe implemente um método diverso. Por exemplo, uma mensagem como:

```
X    aumente:5
```

destinada a fazer o objeto referenciado por X aumentar sua área 5 vezes, poderá ativar rotinas inteiramente diferentes caso X for um Circulo ou um Retângulo, embora produzindo efeito semelhane de aumento de tamanho. Para o usuário a di-

ferença dos métodos usados será transparente. A seção 2.4 discute as implicações dessa característica para a extensibilidade da linguagem.

2.2 Hierarqias de generalização - classes e subclasses.

Smalltalk herdou da linguagem Simula a organização da informação em uma hieraquia de classes e subclasses que constitui uma forma poderosa de se fatorar as propriedades comuns a classes semelhantes. Esse mecanismo é importante por 2 motivos, a saber:

a) permite organizar a informação no computador na forma de abstrações semelhante á utilizada pela mente humana, o que facilita a utilização da linguagem e a compreensão dos programas;

b) evita a repetição (redundância), armazenando em um só lugar informações partilhadas por várias classes, o que resulta em maior integridade, programas mais concisos e manutenção simplificada.

Dada a definição de uma classe C, a linguagem permite definir subclasses Ci de C tais que toda instância de Ci herda automáticamente tanto as estruturas como os métodos de C. A definição de Ci pode acrescentar e/ou redefinir componentes e métodos, onde se caracterizar a diferença para a subclasse. A estrutura de classes forma uma árvore cuja raiz é a classe Objeto. Classes podem ser definidas com o único objetivo de localizar propriedades comuns a várias subclasses, sem que se pretenda que ela tenha instâncias próprias. Nesse caso a classe é dita **abstrata**. Todo objeto ao receber uma mensagem procura inicialmente o método correspondente no repertório de métodos de sua classe. Caso não seja encontrado, a procura passa a ser feita na superclasse de sua classe, e assim sucessivamente até achá-lo ou chegar á classe Objeto, que contém a descrição do protocolo comum a todos os objetos do sistema.

2.3 Uniformidade do sistema Smalltalk-80.

Smalltalk-80 é uniformemente orientada para objetos. Todos os conceitos do sistema são implementados através de alguma classe, até mesmo as próprias classes, que são instâncias da classe Class. Numeros, texto, datas, coleções de objetos, estruturas de dados, arquivos, processos, métodos de acesso, janelas da tela, desenhos e figuras, simulações, editor de texto, o próprio interpretador, tudo enfim no sistema que pode ser referenciado ou manipulado é implementado de forma uniforme e pode ser acessado através do mesmo mecanismo de troca de mensagens por qualquer objeto do sistema. Smalltalk constitui portanto um ambiente integrado de software onde não existe distinção entre modos normalmente encontrados em sistemas convencionais tais como sistema operacional, programa, dados e utilitários.

2.4 Polimorfismo e extensibilidade.

Ingalls (1986) observa que o paradigma de troca de mensagens entre objetos resolve o problema de explosão de complexidade na utilização de procedimentos polimórficos, que são importantes no caso de linguagens extensiveis. Procedimentos polimórficos são aqueles que trabalham com um ou mais argumentos sem tipo determinado, como por exemplo um procedimento

```
compare (x,y)
```

que retorna **Verdadeiro** caso **x** e **y** sejam iguais, e **Falso** em caso contrário, independentemente do tipo dos argumentos. Dessa forma fica conveniente criar novos tipos de objetos sem precisar alterar na mesma proporção a quantidade de descrições dos procedimentos para manipulá-los. O problema é que para fazer isso em linguagens convencionais é necessário testar o tipo de cada argumento e acionar uma rotina adequada a cada caso. A consequência é uma violação dos princípios de modularidade (a inclusão de um novo tipo obriga a reprogramar varios procedimentos em uso por outros objetos), além de provocar um grande aumento de complexidade em programas maiores.

No modelo de programação por troca de mensagens, como em Smalltalk-80, esse problema pode ser superado de forma satisfatória. Quando um objeto X recebe a mensagem

X compare:y

é ativado apenas o método da classe de X correspondente à mensagem **compare:y**. Os métodos particulares de cada classe não são polimórficos (nem precisam ser) e são programados para trabalhar com tipos determinados, muito embora as mensagens (e as correspondentes descrições de comportamneto para uso externo) possam permanecer comuns a várias classes diferentes. Além disso, a modularidade é preservada, pois a criação (ou remoção) de uma classe implica apenas em criar (ou remover) métodos locais à classe, sem interferir com os objetos e métodos já em uso.

Essa facilidade pode se estender ainda no caso de polimorfismo múltiplo, ou seja, no caso em que, além do objeto receptor de uma mesma mensagem puder variar de tipo, também para cada tipo de receptor os argumentos puderem ser de tipos variados. Uma demonstração de como isso pode ser realizado facilmente em Smalltalk-80 é descrita em [Ingalls86].

3. Contribuições das linguagens orientadas para objetos (LOO's) para a Engenharia de Software.

Nesta seção serão examinadas algumas contribuições que o paradigma da programação orientada para objetos pode trazer para a solução de alguns problemas da Engenharia de Software, mostrando que muitos desses problemas podem resultar do uso de modelos inadequados de programação.

3.1 Re-utilização de software.

Smalltalk (e outras LOO's) estimula de forma natural a reutilização de software. Sendo um ambiente integrado de programação, todos os recursos do sistema estão disponíveis de

maneira uniforme para qualquer aplicação. A partir do momento em que uuma classe é definida e implementada, qualquer aplicação pode utilizá-la diretamemnte, ou definir subclasses especificas para a aplicação, com custo marginal. Por exemplo, basta definir uma vez a classe ArvoreBinária, e qualquer aplicação pode gerar quantas instâncias dessa classe forem necessárias, e utilizar suas propriedades, sem necessidade de escrever uma única linha de código.

Essa facilidade permite abordar um projeto de software de forma semelhante à construção de sistemas físicos: como um objeto complexo constituído pela reunião de diversos objetos pré-fabricados de uso geral, cada qual com estrutura e propriedades bem definidas.

3.2 Confiabilidade, Custos e Manutenibilidade.

Como consequência, os custos de produção de cada classe são divididos entre todas as aplicações que dela se utilizam, com redução considerável do custo por aplicação. Por outro lado, aplicações podem ser implementadas em menor prazo devido à redução do esforço de especificação e programação, e os produtos resultantes apresentam um maior grau de confiabilidade, já que utilizam classes previamente testadas. Por fim, a manutenção, tanto corretiva como evolutiva, fica grandemente simplificada como resultado do uso de componentes padronizados, e pela total modularidade da arquitetura do sistema.

3.3 Facilidades para construção de protótipos rápidos.

Embora as implementações de Smalltalk e outras LOO's ainda não sejam eficientes o bastante para competir com as linguagens convencionais, são por outro lado potencialmente úteis para a construção de protótipos rápidos, para uso na depuração das especificações de aplicações. Sistemas funcionalmente corretos podem ser implementados rápidamente em Smalltalk e colocados à disposição dos usuários para utilização experimental. Modificações nas especificações podem ser facilmente implementadas e testadas realisticamente em muitos

casos. Dessa forma torna-se possivel convergir mais rápidamente para um produto idealmente próximo das expectativas do usuário.

4. Perspectivas para o futuro.

4.1 Paralelismo e Concorrência.

O paradigma de computação através da troca de mensagens entre objetos considera cada objeto como sendo idealmente um processador independente, com memória e procedimentos locais. Esse modelo deverá beneficiar-se dos progressos no campo das arquiteturas paralelas, aliando as vantagens apresentadas nas seções anteriores com implementações eficientes.

Diversos modelos para computação paralela orientados para objetos tem sido desenvolvidos recentemente, abrindo perspectivas promissoras para sua utilização [Yonezawa, Briot e Shibayama86, Ishikawa e Tokoro86, Yokote e Tokoro86].

4.2 Implementações eficientes em máquinas convencionais.

Enquanto isso, implementações eficientes de Smalltalk 80 em máquinas não paralelas tem sido obtidas com relativo sucesso. Samples, Ungar e Hilfinger (1986) conseguiram alto desempenho com uma implementação modificada em um simulador de uma máquina com conjunto reduzido de instruções (RISC). Outras implementações eficientes são descritas em [Caudill e Wirfs-Brock86] e em [Lewis et al.86]

5. Sumário e Conclusões.

Linguagens orientadas para objetos podem constituir uma alternativa válida para resolver a "crise de software", na medida em que permitem reduzir a complexidade envolvida na construção de grandes sistemas de software através da distribuição do esforço de programação por diversas classes independentes de objetos. Modelos de sistemas baseados em objetos são mais próximos da

forma como a mente humana percebe a realidade e portanto mais fáceis de compreender, definir e verificar. Smalltalk-80 é uma linguagem uniformemente orientada para objetos que utiliza como paradigma de programação a troca de mensagens entre objetos, e onde cada objeto é uma instância de uma classe que contém as propriedades comuns de suas instâncias. A re-usabilidade de classes previamente definidas permite conjeturar que essas linguagens podem servir de base para a construção de protótipos rápidos para diversas aplicações.

REFERENCIAS

Birtwistle G. et al. "Simula Begin", Auerbach, Philadelphia, 1973.

Caudill,P.J. e A.Wirfs-Brock "A Third Generation Smalltalk-80 Implementation", em [Meyrowitz86], pp.119-130.

Goldberg,A. e D.Robson "Smalltalk-80 - The Language and its Implementation", Addison-Wesley, Reading, MA, 1983.

Ingalls,D.H.H. "A Simple Technique for Handling Multiple Polymorphism", em [Meyrowitz86],pp. 347 - 349.

Ishikawa,Y. e M.Tokoro "A Concurrent Object-Oriented Knowledge Representation Language Orient 84/K - Its Features and Implementation", em [Meyrowitz86], pp.331 -340.

Kay, A. "FLEX, a flexible extensible language", M.Sc. Thesis, Univ. of Utah, May68 (Univ. Microfilms).

Kay, A "The Reactive Engine", Ph.D. Thesis, Univ. of Utah, Sept.69 (Univ. Microfilms)

Krasner,G.,ed. "Smalltalk-80: Bits of History, Words of Advice", Addison-Wesley, Reading, MA, 1983.

Lewis,D.M. et al. "Swamp - A Fast Processor for Smalltalk

80", em [Meyrowitz86], pp.131-139.

Liskov,B. e S.Ziller "Programming With Abstract Data Types", SIGPLAN Notices, April 1974, pp.50-59.

Meyrowitz,N.,ed. "OOPSLA'86 - Object Oriented Programming Systems, Languages and Applications - Conference Proceedings - Sept23 - Oct2, 1986, Portland, Oregon (SIGPLAN Notices vol.21, no.11, Nov.86).

Robson,D. "Object-Oriented Software Systems", Byte, August 1981, pp.74-86.

Samples,A.D.,D.Ungar e P.Hilfinger "SOAR -Smalltalk Without Bytecodes", em [Meyrowitz86], pp.107-118.

Ungar,D. e D.Patterson "Berkeley Smalltalk - Who Knows Where the Time Goes?", em [Krasner83], pp.189-206.

Yokote,Y. e M.Tokoro "The Design and Implementation of ConcurrentSmalltalk",em [Meyrowitz86],331-340

Yonezawa,A.,J.P.Briot e E.Shibayama "Object-Oriented Concurrent Programming in ABCL/1", em [Meyrowitz86], pp. 258-269.