Artigo sobre paradigmas de programação

Cauã Ribas Devitte

Curso de Ciência da Computação - Universidade do Vale do Itajaí
cauaribasone@gmail.com, cauaribas@edu.univali.br

1. Resumo

A programação é fundamental para a computação. É tanto uma ferramenta prática que utiliza o poder da computação, quanto uma fonte de estimulação intelectual. Linguagens de programação diferentes têm recursos diferentes. As linguagens de programação são a ponte entre o humano e o computador. As linguagens de programação são executadas por computadores, que funcionam com uma lógica inflexível e muito diferente da comunicação humana.

Muita discussão sobre linguagens de programação gira em torno da sintaxe, porque este é o aspecto mais facilmente visível da linguagem e aquele que os programadores principalmente manipulam. Um grande desafio na programação, no entanto, é mapear a sintaxe, porque o comportamento do computador é difícil de ver e controlar. Além disso, o objetivo final é criar programas que se comportam de uma maneira particular.

2. História

Diferentes abordagens sobre programação se desenvolveram ao longo do tempo e foram reconhecidas como tal na época ou retroativamente. Uma das primeiras abordagens conscientemente reconhecidas é a Programação Estruturada, que existe desde meados da década de 1960. O próprio conceito de um "paradigma de programação" remonta pelo menos a 1978. A palestra do Prêmio Turing de Robert W. Floyd intitulada "Paradigmas de Programação" cita o conceito de paradigmas usado por Thomas Kuhn em seu livro The Structure of Science. Revolution (1962). As primeiras linguagens de programação não tinham um paradigma de programação bem definido e, às vezes, os programas faziam uso extensivo de instruções goto. O uso intenso dessas instruções resulta em um "código espaguete" desajeitado. Isso levou ao desenvolvimento de um paradigma de programação estruturada que proibia o uso de instruções goto e permitia apenas o uso de construções de programação bem definidas.

3. Paradigmas

Para estruturar o estudo das linguagens, muitos autores têm utilizado a noção de “paradigma”. Paradigmas são supostamente grupos que diferenciam uma classe de linguagens semelhantes de outras de alguma forma de alto nível, geralmente focados em recursos que exibem comportamentos comuns. Os autores listam convencionalmente alguns paradigmas principais: imperativo, orientado a objetos, funcional e lógico, e outros autores tendem a adicionar um ou mais scripts, Web, banco de dados e/ou reativo.

Em linguagens reativas, o programa expressa dependências, mas a linguagem lida com a atualização dos valores das variáveis na presença de mutação. Da mesma forma, na lógica e paradigmas de banco de dados, os programas expressam dependências lógicas entre elementos de dados, mas a determinação das respostas é feita por meio de um algoritmo codificado na linguagem (hard-coded).

4. Programação Orientada a Objetos

OO é um termo amplamente usado e cheio de ambiguidade. OO depende de objetos, que são valores que combinam dados e procedimentos. Os dados geralmente estão ocultos ("encapsulado") do mundo exterior e acessível apenas a esses procedimentos. Esses procedimentos têm um argumento especial, cujos dados ocultos eles podem acessar e, portanto, são métodos chamados, que são invocados por meio de despacho dinâmico. Isso é comum a todas as linguagens OO, mas além disso elas diferem amplamente:

A maioria das linguagens OO tem um objeto distinto do qual os métodos dependem, mas algumas em vez disso, tem multimétodos, que podem despachar muitos objetos ao mesmo tempo.

Algumas linguagens OO têm uma noção de classe, que é um modelo para fazer objetos. Nessas linguagens, é vital que os programadores entendam a distinção classe-objeto. No entanto, muitas línguas consideradas OO não possuem classes. A presença ou ausência de classes leva a muitos padrões diferentes de programação.

A maioria das linguagens OO tem uma noção de herança, em que um objeto pode se referir a alguma outra entidade para fornecer comportamento padrão. No entanto, existem grandes variações de herança: a outra entidade é uma classe ou outro objeto (prototípico)? Pode se referir a apenas uma entidade (herança única) ou para muitos (herança múltipla), e se o último, como as ambiguidades são resolvidas? É o que se refere fixo ou pode mudar conforme o programa opera?

Algumas linguagens OO têm tipos e a função dos tipos na determinação do comportamento do programa pode ser sutil e pode variar um pouco entre os idiomas. Em geral, todas essas variações de comportamento são agrupadas como OO, embora eles levam a designs de linguagem significativamente diferentes e comportamentos correspondentes, e não são mesmo exclusivo para ele (por exemplo, fechamentos funcionais também encapsulam dados). Assim, uma frase como "objetos-primeiro" pode, em princípio, significar dezenas de estruturas curriculares totalmente diferentes, embora na prática pareça referir-se a currículos construídos em torno de objetos encontrados em Java.

A Programação Funcional também aparece com frequência na literatura educacional, popularizada pelo livro seminal de Abelson e Sussman (1985). Na Programação Funcional, os programadores fazem pouco ou nenhum uso de atualizações imperativas. Em vez disso, os programas consomem e produzem valores, e a programação é vista como o arranjo de funções para compor e decompor valores (alguns até apelidaram Programação Funcional como
"programação orientada para o valor"). Devido à falta de mutação, problemas de aliasing são essencialmente inexistentes. Programação Funcional é caracterizado por mais duas

características: a capacidade de passar funções como valores, o que cria operações de nível muito mais alto do que os loops tradicionais (um problema que se manifesta na composição plana) e chamadas finais, que criam soluções recursivas semelhantes a loops tão eficiente quanto os loops, permitindo assim a recursão como uma forma primária e generalizável de looping. As duas principais variações na Programação Funcional são se o idioma é digitado ou não e se a computação é ansiosa ou preguiçosa, cada uma das quais leva a uma diferença significativa no estilo de programação.

5. Mais sobre paradigmas

Embora os paradigmas estejam em uso há muito tempo, vale a pena perguntar no que eles contribuem para o nosso entendimento. Em primeiro lugar, deveríamos ver os paradigmas como classificadores que agrupam as linguagens em seções exclusivas? Eles certamente são interpretados dessa forma por muitos leitores, mas duas coisas devem:
Ser imperativo não impede ser reativo. Portanto, estes não podem ser vistos como independentes. A reatividade simplesmente dá uma interpretação adicional para uma atualização imperativa; além disso, as linguagens reativas funcionais têm comportamento reativo sem características explicitamente imperativas, e a reatividade também pode ser adicionada a objetos. Em outras palavras, enquanto OO e Programação Funcional são declarações sobre a organização do programa, a reatividade é uma declaração sobre comportamento do programa na atualização. Estas são questões essencialmente ortogonais, permitindo a reatividade a serem adicionadas aos idiomas existentes.

As linguagens não se organizam em taxonomias hierárquicas como as plantas e os animais; Eles são entidades artificiais que podem ser livremente reproduzidas através de supostas fronteiras. Autores de linguagens podem escolher entre várias seções diferentes ao criar linguagens e, de fato, as línguas tradicionais modernas são geralmente uma mistura de muitas dessas seções. Até linguagens OO arquetípicas como Java agora têm recursos de programação funcionais.

Além disso, paradigmas às vezes combinam sintaxe e comportamento. Por exemplo, alguns os autores agora pensam em linguagens visuais baseadas em blocos como um paradigma; no entanto, os blocos são puramente uma questão de sintaxe e construção do programa, enquanto os outros paradigmas são sobre comportamento.

De fato, existem interfaces de bloco para imperativo, OO e Programação Funcional. Assim, não está claro se os blocos deveriam até ser listados como um paradigma; o que destaca ainda mais a confusão que esse termo cria.

Outra fonte de confusão é se os “paradigmas” são declarações sobre linguagens de programação ou sobre estilos de programação. Por exemplo, pode-se argumentar que eles são programação em “estilo PF” em uma linguagem que normalmente não é considerada “funcional”.

Essas afirmações têm pouca validade, mas devem ser vistas com algum ceticismo. Por exemplo, um programador que está passando ponteiros de função está simulando um nível superficial de programação funcional, mas na ausência de construção de fechamento automático e coleta de lixo correspondente, esta é uma simulação fraca e muitas vezes insatisfatória.

De forma similar, a falta Tail-Call Optimization (TCO)
do Python torna muitos padrões PF naturais inutilizáveis na prática. No entanto, a possibilidade de tais simulações torna ainda mais difícil entender o que paradigmas são.

Outra fonte de confusão é entre a linguagem e o ambiente operacional. Em um programa “batch”, o programa tem início e fim bem definidos. Pode pausar periodicamente para aceitar a entrada, mas o programa determina quando isso acontece. Em contraste, em um evento orientado programa, quem manda é o ambiente operacional; cada evento (seja um pressionamento de tecla, um toque na tela, uma solicitação da Web, a chegada de um pacote de rede ou o tique-taque de um relógio) causa alguma parte do programa a ser executado em resposta. Depois de responder ao evento, o programa (geralmente) retorna para quiescência, e "acorda" no próximo evento. Tal programa não tem um bem definido começo ou fim e, em princípio, dura para sempre. O desafio do programador é organizar como estado é transferido entre os eventos. Isso pode ser feito de várias maneiras: imperativamente, usando objetos, funcionalmente, reativamente, e assim por diante. Assim, a programação orientada a eventos é outra noção transversal que é independente e ortogonal à organização do programa, ao contrário, é uma declaração sobre a relação do programa com seu ambiente operacional.

Falta uma clara definição de “paradigma”, portanto, não está totalmente claro se a uniformidade é um deles.

Assim, embora os paradigmas sejam amplamente utilizados por autores e na literatura, alguns autores, especialmente na comunidade de pesquisa de linguagens de programação, questione seu uso. Nossos exemplos e pontos de discussão ilustram por que um foco em propriedades e características comportamentais fornece um enquadramento mais significativo.

A educação na programação, um tipo de programação imperativa, estrutura os programas como redes centradas no ser humano, como ensaios de hipertexto. A documentação é essencial para o programa, e o programa é estruturado de acordo com a lógica da exposição em prosa, não a conveniência do compilador. Um paradigma de programação declarativa foi desenvolvido independentemente da ramificação imperativa. Essas linguagens informam ao computador qual é o problema, não como resolvê-lo. O programa é estruturado como um conjunto de propriedades para descobrir o resultado esperado e não como etapas a seguir. Dado um banco de dados ou conjunto de regras, o computador tentará encontrar uma solução que corresponda a todas as propriedades necessárias. O progenitor das linguagens declarativas é a família das linguagens de quarta geração SQL, linguagens funcionais e programação lógica. A programação funcional é um subconjunto da programação declarativa. Os programas escritos usando esse paradigma usam funções, que são blocos de código destinados a se comportar como funções matemáticas. As linguagens funcionais desencorajam as atribuições de alterar o valor das variáveis e, em vez disso, fazem uso intenso de recursão. O paradigma de programação lógica vê a computação como raciocínio automatizado sobre um corpo de conhecimento. Fatos sobre o domínio do problema são expressos como fórmulas lógicas, e o programa é executado aplicando regras de inferência às fórmulas lógicas até que a resposta para o problema seja encontrada ou que o conjunto de fórmulas seja contraditório. A programação simbólica é um paradigma para escrever programas nos quais as expressões matemáticas e os componentes do programa podem ser manipulados como dados. Assim, ele efetivamente modifica e parece "aprender" o

próprio programa, tornando-o adequado para aplicações como inteligência artificial, sistemas especialistas, processamento de linguagem natural e jogos de computador. As linguagens que suportam esse paradigma incluem Lisp e Prolog. Um programa com construções de programação diferenciáveis permite que todo o programa seja diferenciado, geralmente por diferenciação automática.

Referências

Friday Joseph Agbo, Solomon Sunday Oyelere, Jarkko Suhonen, and Sunday Adewumi. 2019. A Systematic Review of Computational Thinking Approach for Programming Education in Higher Education Institutions. In Koli Calling 19: Proceedings of the 19th Koli Calling International Conference on Computing Education Research. 12 (1-10).

Abelson, Harold, and Sussman, Gerald Jay. 1985. Structure and Interpretation of Computer Programs. Cambridge, MA: MIT Press. Altadmri, A., and Brown, N. C. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. Pages 522–527 of: Proceedings of the ACM Symposium on Computer Science Education (SIGCSE). Armoni, M., Meerbaum-Salant, O., and Ben-Ari, M. 2015. From Scratch to 'real' programming. Transactions on Computing Education, 14(4).

Bailie, Frances, Courtney, Mary, Murray, Keitha, Schiano, Robert, and Tuohy, Sylvester. 2003. Objects First - Does It Work? Journal of Computing Sciences in Small Colleges, 19(2), 303–305. Bainomugisha, Engineer, Carreton, Andoni Lombide, Cutsem, Tom van, Mostinckx, Stijn, and Meuter, Wolfgang de. 2013. A Survey on Reactive Programming. ACM Computing Surveys, 45(4), 52:1–52:34.

Marina Silva and Ana Paula Ferreira. 2022. Linguagens visuais para o ensino de programação: uma revisão da literatura com foco em paradigmas de programação. In *Anais do II Simpósio Brasileiro de Educação em Computação*, abril 24, 2022, Online, Brasil. SBC, Porto Alegre, Brasil, 18-28.

Shriram Krishnamurthi and Kathi Fisler 2019. Programming Paradigms and Beyond. Available Online at https://cs.brown.edu/~sk/Publications/Papers/Published/kf-prog-paradigms-and-beyond/paper.pdf