5 ドメイン専用言語と言語指向プログラミング

応用

市川和央（東京大学）

言語指向プログラミング

ドメイン専用言語（DSL）は特定の目的に特化した言語のことで，ソフトウェア開発のさまざまな場面で利用されている．たとえば，Web ページを記述するのには HTML，データベースを操作するのには SQL が用いられる．DSL は C 言語や Java のような汎用言語と比べて機能が制限されている代わりに，目的コードが簡潔で読みやすくなるという利点がある．ちなみに，DSL はプログラミング言語に限った話ではない．たとえば，数式は数学のための DSL と言えるし，身内でしか通用しないような略語なども一種の DSL と言える．

DSL を活用した開発スタイルとして，言語指向プログラミング（LOP）というものがある．LOP ではまずはじめに開発に利用する DSL 群を作成し，それを利用してソフトウェアを開発する．ソフトウェアの大部分を DSL を利用して記述するため，ロジックが整理されていて読みやすくなり，保守が容易になるという利点がある．LOP は特に多数のエンジニアによる開発で威力を発揮する．大部分のエンジニアは機能の制限された DSL 上で開発を行い，一部の上級エンジニアのみが汎用言語で開発を行うことで，バグの入り込む余地を小さく絞ることができる．

LOP の問題点は DSL の開発コストがかなり大きい点である．DSL もプログラミング言語の一種なので，コンパイラまたはインタプリタが必要となる．また，実際に LOP で開発をすることを考えると，エディタなどの開発環境も必要となるだろう．LOP を実際に行うためには，DSL の開発を支援する何らかの仕組みが必要となる．

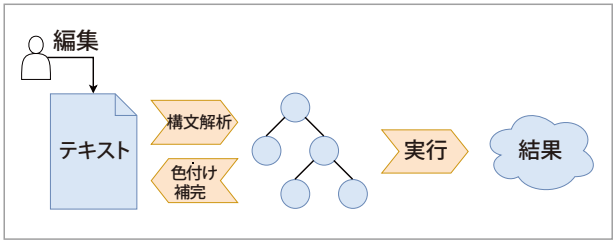

図 -1　プログラミング言語処理系とエディタ

言語ワークベンチ

言語ワークベンチは DSL を開発するための統合開発環境で，DSL を実装するとエディタなどの開発環境が同時に作成される．言語ワークベンチのアイディアは単純で，言語の実装の一部を開発環境の実装に再利用するというものである．**図 -1** はプログラミング言語処理系とエディタの動作を簡単に書いたものだ．言語処理系はテキストを構文解析しそれを実行する．エディタは構文解析の結果をシンタックスハイライトなどの形でテキストにフィードバックする．いずれも構文解析部分は共通しているため，再利用することができる．このような再利用を言語実装のさまざまな点で行うことで，DSL の開発環境を半自動的に作成できる．言語ワークベンチでは，DSL の実装の各部品の再利用を可能とするために，DSL の実装そのものを DSL で行う．たとえば Spoofax という言語ワークベンチでは，SDF3 という DSL で文法を，NaBL という DSL で変数名や型名の名前解決および型検査を，Stratego という DSL でシンタックスシュガーを，DynSem という DSL で操作的意味論を記述する．これらの DSL を用いることで，宣言的に DSL を実装することができる．

Meta Programming System（MPS）は投影エディタという非常に面白い方式を採用している言語ワー

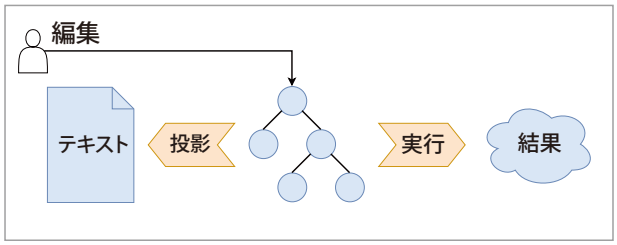

図 -2　投影エディタを利用した開発

クベンチだ．投影エディタでは，ユーザはコード補完コマンドを打つことで DSL のプログラムを記述する．コード補完コマンドを打つと，その部分で使うことのできる構文の一覧が表示される．ユーザはそれらの中から適切なものを選ぶことでコードを記述する．この方式のユニークな点は，**図 -2** のように，ユーザがテキストではなく構文木を記述する点である．エディタ上に表示されるプログラムは，構文木をテキストとして投影したものとなっている．プログラム中の空欄部分をコード補完によって埋めることで，ユーザは構文木のノードを選択しているのである．この方式は複雑な構文解析や名前解決の必要がなく，構文木，投影関数，およびその意味論を実装するだけで DSL 処理系とエディタを実装することができる．

言語内 DSL

ほかの DSL 開発手法として，言語内 DSL（埋め込み DSL とも言う）を開発するというものがある．言語内 DSL は DSL を汎用言語のライブラリとして実装したものである．単なるライブラリなので，ユーザは気軽にインポートして利用することができる．ベースとなる汎用言語（ホスト言語）の開発環境がそのまま使えるのも利点の 1 つである．

本稿では，言語内 DSL の作り方を浅い埋め込み（shallow embedding），深い埋め込み（deep embedding），全埋め込み（full embedding）に分類する．**図 -3** にこれらの概略を示す．浅い埋め込みでは実行時に DSL プログラムの実行結果を生成する．要するに，DSL のように見える普通のライブラリである．深い埋め込みでは実行時に DSL プログラムの構文木を生成し，それを実行する．全埋め込みでは実行時に DSL のソースコード文字列を生成し，それを構文解析して実行する．

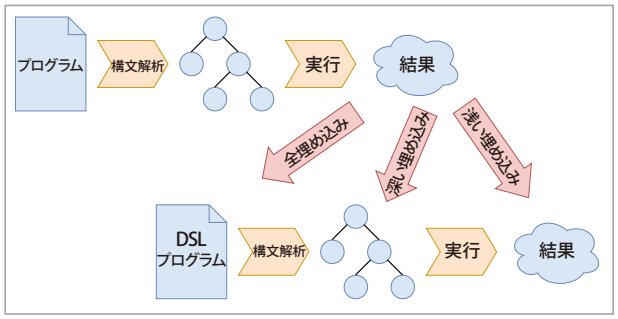

図 -3　言語内 DSL の分類

浅い埋め込みの DSL は実装が容易だが，DSL 固有の最適化を行うことができない．深い埋め込みは DSL 固有の最適化が可能だが，実装はやや煩雑になる．これらの DSL は文法がホスト言語によって制限されるが，全埋め込みであれば自由な文法の DSL を設計できる．しかし，全埋め込みの DSL は実装コストが高い上，ホスト言語の開発環境をほとんど活用できない．これらの手法は一長一短で，それぞれの短所を補うような研究が進められている．

本稿では，DSL の開発コストを低減する開発手法として，言語ワークベンチと言語内 DSL という 2 つの手法を紹介した．このような取り組みは，コードの可読性を向上させ，保守を容易にするのに役立つ．

（2017 年 7 月 26 日受付）

市川和央　ichikawa@csg.ci.i.u-tokyo.ac.jp

東京大学 情報理工学系研究科 学術支援職員，プログラミング言語の構文拡張に関する研究に従事，まもなく博士号取得予定．