プログラミング言語 15－4
（1988. 2. 19）

ソフトウェア開発環境とその言語モデル

山岡 健志, 斎藤 信男

慶応義塾大学
理工学部 数理科学科

概要

高品質のソフトウェアを効率よく開発していくためには、高機能のソフトウェア開発環境が必要になる。ソフトウェア開発環境の実現を計るためには、いくつかのアプローチが考えられる。ここでは、その一つであるモデルベーストアプローチの具体例として、 SDA(Software Designer's Associates)を提示する。

一方、ソフトウェア開発環境と言語とは多くの接点で関連を持つが、その一つは開発環境を一つの言語モデルと解釈することである。ここでは、SDAのモデルを言語解釈系と見なすことを論ずる。また、具体的なソフトウェア開発環境の実現、そこに於けるソフトウェアプロセスの記述などに使う言語は、重要な役割を果たす。ここでは、それらの言語に関する特徴などを論ずる。

SOFTWARE DEVELOPMENT ENVIRONMENT AND ITS LANGUAGE MODEL

Takeshi YAMAOKA, Nobuo SAITO

Department of Mathematics
Faculty of Science and Technology
Keio University
3-14-1 Hiyoshi, Kohokuku, Yokohama, 223, Japan

ABSTRACT

It is very important to realize an elaborated software environment with high performance and facilities. There are several approaches to realize such a development environment, and the authors show the model based approach used in SDA(Software Designer's Associates) as an concrete examples of the approaches.

There are several inter-relationships among software development environment and languages. One is that the software development environment can be considered as an interpreter of a language. As an example, the authors discuss the SDA model as an interpreter of its description language. The languages used to implement the environment and the langauges used to describe the software processes in the environment are also very important. This report shows several examples of such languages with their characteristics.

1. ソフトウェア開発環境へのアプローチ

ソフトウェア開発環境は、高品質のソフトウェアを得るためには必須のシステムである。この追求は現在多くのところで行われているが、その方法はいくつかのものがある。これは、どれだけ構成要素の扱いに自由度を与えるのかによって異なってくる。たとえば、次のようなアプローチが考えられる。

(1)ツールインテグレーション(Tool Integration)

これは、ソフトウェア開発の具体的な作業で使われるソフトウェアツールを重視し、その統合化を計ることを目的とした開発環境を実現していこうという方法である。具体的には、Gandalf[1], APSE(Ada Programming Support Environment)[2]などが挙げられる。

(2)プロセスインテグレーション(Process Integration)

具体的なソフトウェア開発においては、そのプロセスが重要な役割を果たす。そこで、ソフトウェア開発環境も当然ソフトウェアプロセスを重要視してその実現をはかるために必要な機能を全て支援するようにしなければならない。ソフトウェアプロセスの支援にとって必要なツールがあればそれを生成することまで環境内で行いたい。プロセスプログラミングとそれに基づいたArcadiaプロジェクト[3]は、この方法の代表的な具体例である。

(3)モデルベーストアプローチ(Model Based Approach)

これは、プロセスも含めたソフトウェア開発環境のモデルを想定し、その実現を計る方法である。ここでは、たとえばソフトウェアプロセス自身の開発や設定までも含めて環境内で支援できなければならない。この具体的な例として、SDA (Software Designer's Associates)[4]がある。

(4)ヒューマンベーストアプローチ(Human Based Approach)

これは、ソフトウェア開発の主体的な担い手であるエンジニアを中心に考える方法であり、その経験や応用分野に応じた環境モデルの設定、そのインスタンスの実現なども含めた機能を支援する方法である。そこでは、人間の知識および経験の蓄積や利用が十分に行われる。この具体的なものはまだ多くは見られないが、そのような方向を示しているものとしてDraco[5]がある。

2. モデルベーストアプローチ

ここでは、モデルベーストアプローチについて具体的な例を示しながら論ずる。ソフトウェア開発環境の構築要素を求め、環境のアーキテクチャを追求するためには、環境のモデルを抽象的に設定しその要素を考慮する方式が役に立つ。SDAは、ソフトウェアのデザインプロセスを主としてあらゆるソフトウェア開発の支援をするための開発環境構築プロジェクトであり、産学共同、日米共同を実地に行っている。

このプロジェクトは、最初から開発環境のモデルを提案し、それに従って環境アーキテクチャを追求しその実現を計ろうというものである。その環境モデルを図2.1に示す。ここでは、プロダクト(Product)、プロセス(Process)およびツールの集合(Tool Set)を環境に現れる基礎概念とし、それらの相互作用を管理し制御するのがソフトウェア開発環境であると見なす。

その概念から実際のシステムの実現までは、抽象化の階層を設定する。すなわち、概念モデル(Conceptual Models)、意味モデル(Semantic Models)および実現モデル(Realization Models)の三層からなる。概念モデルは、開発環境のユーザとの接点であり、ユーザが考慮すべき基本概念を示し、プロダクト、プロセス、およびツールの集合を含む。意味モデルは、その概念を詳しく説明するためのモデルであり、解析モデル、ツールの相互作用、および基本支援系(Substrate)を含む。基本支援系は、情報倉庫、通信系、ユーザインタフェースなどいわゆるオペレーティングシステムのようなシステムを指す。実現モデルでは、解析管理システム、ソフトウェアプロセス管理システム、およびオブジェクト管理システムからなる。この環境では、プロダクト、ツールをはじめ、ソフトウェアプロセスを記述するプロセスプログラムもオブジェクトとして扱えるので、統合的なオブジェクト管理システムが必要になる。

SDAでは、その環境は対象となる開発プロダクト、開発グループ、開発エンジニアなどにより、採用するソフトウェア開発方法論、開発プロセス、開発ツールなどが異なってくる。そこで、最初の開発すべき環境はあくまでも生成的なもの(generic SDA)であり、その用途により環境のエンジニアやユーザがカスタマイズして特定な環境(specific SDA)を作成して利用する。その意味で、SDAの環境の構築要素であるプロセスやツールは全て生成的に実現されていなければならない。これは、Gandalfプロジェクトが掲げていた目標とも一致する。

ソフトウェア開発は、人間であるエンジニアが行う作業が基本的な要素となっている。この作業は、エンジニアの経験を深め優れたエンジニアに成長していく過程でもある。このような過程は、一つの知識として同一のエンジニアのみならず他のエンジニアにとっても有効に役立たせることが出来る。このような過程からどの様にして有効な知識を得るかということは簡単な問題ではないが、少なくともソフトウェア開発過程におけるあらゆる現象や関連要素を正確に記述しておく事がまず必要になる。その意味では、SDAの様に環境モデルをまず設定し、それに沿ってあらゆる事を考慮し記述していくことは、ソフトウェア工学に於ける知識の利用、知識工学との接点を追求するためのきっかけを与えるものである。

3. 開発環境モデルと言語モデル

計算機における環境とは、実質的に用いることができる資源と、その使い方に関する情報である。この環境を明示的に表現するために、環境を記述する言語モデル、すなわち、開発環境モデルを構築するための言語モデルを提案する。

(1)言語モデル

環境を構築する要素には、参照することができる資源と、資源の中でも特に作業を支援するものとしてのツールセットを考えることができる。そして、さらに、資源をいかに使うかを示すものが必要である。ここで提案する言語モデル(図3.1)では、このそれぞれを、プロダクト、ツールセット、プロセスとし、これらを言語で表現し、解釈系を考える。

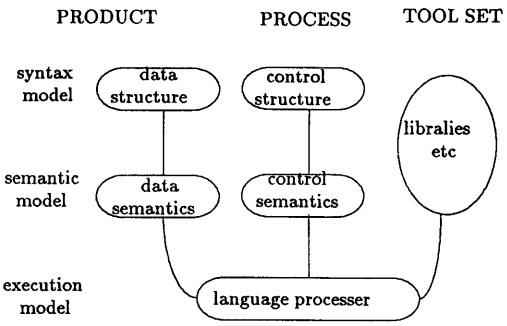

図3.1 言語モデル

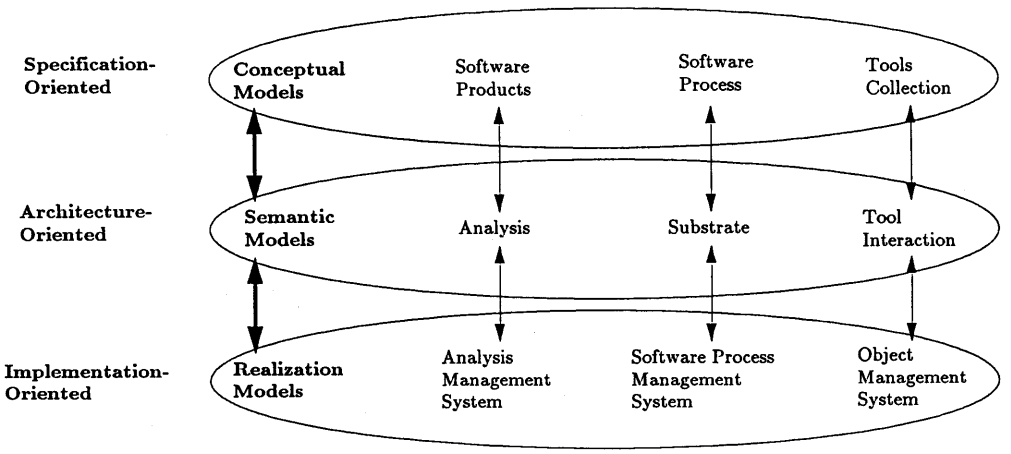

図2.1 SDA における環境モデル

UNIX を具体的な例として考えると、環境を定義するものとして、シェルがある。シェルにおいて、資源はファイルであり、ツールセットはコマンド群に対応するであろう。しかし、ファイルやコマンドをいかに使うかは、ユーザの個人的な知識によっている。プロセスの記述はシェルスクリプトを挙げることができる。しかし、これらは、暗黙的な了解のもとで使われていて、明示的な環境の表現はない。

(2)プロダクト

プロダクトは資源の指定である。環境の中で参照することのできる資源を規定する。また、ある作業による中間成果物、最終成果物、結果などを示す。従って、ある作業に際しての対象物であり、材料と成果の表現になる。

ここでは、ファイルや OS の機能など、あらゆるオブジェクトを考え、表現すべきである。言語としては、扱う対象としてのデータ構造であり、表現として、変数と考えることができる。

(3)ツールセット

ツールセットはツールの集合であり、ツールとは、作業を支援するものである。これは、ある環境から起動することのできるツールの規定である。言語としては、ツールは手続きや関数に対応し、その集合は、ライブラリと考えることができる。

(4)プロセス

プロセスは作業の過程を示すものである。言語としては、制御構造であり、変数や手続きを指定して、材料から目的の結果を得る動作である。このプロセスをプロダクト、ツールセットとともに、言語によるプログラムとして明確に残すことで、環境における作業の履歴を残せることも有益である。言語として、動作の階層化や抽象化を考えることにより、最上層の部分で、目的達成のための動作の流れを記述し、下の層では、中間物のための記述をすることになる。これらの記述が、環境としての基本的な知識となり得る。

(5)記述言語

今まで述べてきた特徴を表現するため、モジュールベースの言語を提案する。基本的には、Modula-2 言語[6]におけるモジュールの概念と同様であり、オブジェクトの参照の関係を示し、インタフェイスの意味を持つ定義モジュールと、その定義モジュールで規定されているオブジェクトを用いて、手順を示す実現モジュールが別に存在する。プロダクトとツールセットの記述は、定義モジュールとして、使えるオブジェクトや関数を定義する。プロセスの記述は、実現モジュールとして、その時の環境のもとで行なうことのできる動作を記述する。また、Modula-2 でいうプログラムモジュールは、最終物のための流れの記述であり、それ以外のモジュールでは、中間物のための記述となり、これが作業のための知識となり、環境としての要素になる。

```
DEFINITION MODULE print;
   FROM text    IMPORT textext;     (* type *)
   FROM printer IMPORT impprinter(imp);  (* proc *)
   FROM tools   IMPORT tex2dvi(tex,dvi); (* proc *)
   FROM tools   IMPORT dvi2imp(dvi,imp); (* proc *)
   FROM file    IMPORT remove(FILE...);  (* proc *)
.....

   EXPORT print(tex);
   EXPORT print(roff);
END print.


IMPLEMENTATION MODULE print;
   VAR textfile : textext;
       dvifile : dvi;
       impfile : imp;
.....

   PROCEDURE print(file:tex);
       dvifile = tex2dvi(file);
       impfile = dvi2imp(dvifile);
       impprinter(impfile);
       remove(dvifile, impfile);
   END print;
   PROCEDURE print(file:roff);
.....

BEGIN
END print.
```

例題プログラム

4. 開発環境構築のためのアーキテクチャ

UNIX などの環境を持つワークステーションにおいては、ファイルやツール、さまざまな機能の資源などが提供され、ユーザはこれらを自由に用いることができる。これらの利用は、ユーザに任されている。この汎用性のため、利用の知識を持つユーザでないと使いこなせない。しかし、汎用性を取り去り統一的にしてしまった

のでは、使いにくくなる可能性が高い。このために、環境を設定、構築するための機構が必要である。これにより、明示的な表現を伴った環境を提供する。 3章で述べたように、環境は、資源、ツールの規定と、作業についての知識である。これで、ある目的のために必要な環境を構築し、現有の環境を統合して、仮想的な環境を構築することを目的とする。

ここでは、このようなアーキテクチャとして、 SDA に提案した環境構築のためのアーキテクチャ(図4.1)を挙げる。これは、3層に分かれ、最下層は、現実の世界であり、現有の個々の環境の集合である。例えば、複数のワークステーションのそれぞれの環境の集合がある。もちろん、それぞれにおいては、それぞれで与えられた環境を用いることができる。最上層は、必要とされる機能を備えた環境である。与えられる現有の環境と必要とされる環境により、それぞれ構築しなおす必要がある。中間層は、上下の層のインタフェイス部分であり、最下層からの機能の抽象化、また、最上層で必要な機能の提供を実現する。

それぞれの環境においては、シェルアプローチをとり、それぞれにおいて環境としての記述を行なう。このアーキテクチャをある目的に応じて実現することにより、目的にそくした環境構築を行なうことができ、しかも環境の表現がされることもあり、目的や現有環境の変化によっても、新たな環境を構築できるであろう。

つぎに、図4.1のアーキテクチャについて説明する。これは、UNIX の環境に対し、これらの資源やツールを、統合化、仮想化して、デザインにおける支援を目的とし、プロセスモデルの記述を行なおうというものである。

(1)SDA Environment

この層は、SDA の環境として、プロダクト、プロセス、ツールの各サーバを提供する。 SDA のユーザはデザインに関するプロセス記述を、プロダクトとツールの指定とともに行なう。基本的には、プロセスはデザインモデルの記述であり、プロダクトは生成物を管理するための記述であり、ツールは起動するツールの規定である。

(2)Typed Object Environment

この層は、最も重要な部分になるであろう。 SDA Environment に必要な機構をここで定義する。すなわち、さまざまな管理機構を定義する。 UNIX Environment における基本構成要素を、型を持ったオブジェクトとして、統一的に扱い、これらを用いて、上の層でのオブジェクトの扱い方を定義する。例えば、生成物としてのプロダクトの管理(バージョン管理など)や、デザインプロセスにおける知識の保存と参照法などの定義を記述する。この部分での必要な機能については、かなりの

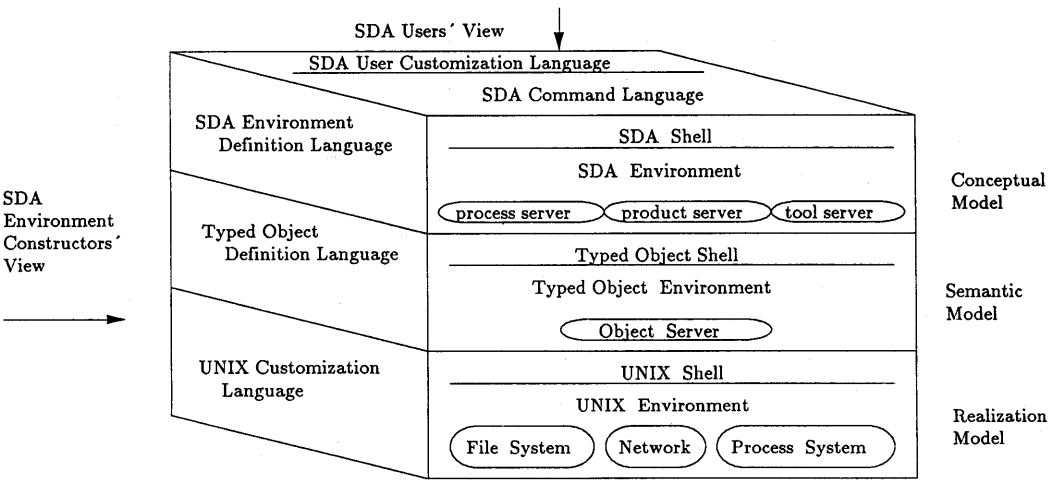

図4.1 環境構築のためのアーキテクチャ

議論が必要である。

(3)UNIX Environment

UNIX は、さまざまな機能や資源を提供する。ファイルシステム、ネットワーク機能、ユーザインタフェイス、コマンドなどである。また、分散環境を考えることもできる。これらの機能や資源は、環境構築のための基本構成要素である。この上の環境においても、この基本構成要素を抽象化して扱うのである。環境構築記述言語により、オブジェクトの関係とそれに対する個々の動作が記述される。また、シェルアプローチとして、必要な要素を上の環境に、明示的に提供する。

(4)環境構築記述言語

各層において、環境を記述するための言語が必要である。ここでは、3章で述べた、モジュールベースの言語が適当である。上の層に提供する要素を、外に出す定義モジュール内で宣言することにより、シェルアプローチが実現できる。全体的に見れば、モジュールの階層構造と、そのスコープルールに従うことになる。また、プログラミング言語として、抽象化、詳細化、オブジェクトの相互関係、また、それに対する操作を統一的に記述できる。

(5)環境構築プログラマ

今まで述べてきたことは、環境構築の機構とアーキテクチャである。これに、プログラミング言語を適用し、環境をプログラムとして表現する。このためには、さまざまな管理機構や環境設定のために、環境をソフトウェアプログラムと同様に設計し、プログラミングを行なうプログラマが必要になるであろう。。

5. ソフトウェア開発環境の将来

この報告では、ソフトウェア開発環境をなるべく一般的に捉え、その機能の柔軟化を目指して実現することを提案している。ソフトウェア開発そのものを一つの言語モデルとして捉え、その解釈系を環境として実現することを目指している。

このように、ソフトウェア開発環境は単なるツールの集合として捉えるのではなく、ツールを駆動し利用する各種作業、ツールを適用すべき対象としての各種プロダクト、それらの間のいろいろな形態の相互作業などを、一般的、統合的に捉えようとしている。

このようにソフトウェア開発環境をより一般的にとらえ、そのあらゆる側面を正確に記述していくことは、その記述を有効に利用しようという方向に発展していく可能性がある。これは、いわゆる知識の利用という方向を目指すものである。たとえば、ソフトウェアの設計過程におけるエンジニアの意志決定の履歴は、以後の設計作業にとって、大いに有効な情報として利用できる。また、開発方法論、プロセスプログラム、ツール生成など実際のソフトウェア開発にとって重要な問題を、知識の利用によってより有効で正確に解決していくことが可能となる。これは、生成的な環境を開発の対象に応じて特定化していく過程を制御する際に有効に役立てることができる。

いずれにしろ、このようにエンジニアの能力や経験に応じて開発環境を管理し制御していけるわけであり、ソフトウェア開発の知識を利用したよりメタな環境は、エンジニアあるいは人間を中心とした設計になっている。この意味では、将来の開発環境は、ヒューマンベーストアプローチになっていくはずであり、またそうでなければならないのである。

謝辞

本報告の作成に当たり、お手伝いいただいた、慶応大学山口晃君、また、きっかけともなった SDA プロジェクトの諸氏に感謝致します。

参考文献

[1] A.N. Habermann et al.,The GANDALF System Reference Manuals, Department of Computer Science, Carnegie-Mellon University, 1986.

[2] ADA: STONEMAN : Requirements for Ada Programming Support Environments, Department of Defense, 1980.

[3] R.N.Taylor, L. Clarke, L.J.Osterweil, J.C. Wileden, and M.Young, Arcadia: A Software Development Environment Research Project, Proceedings of the Second International Conference on Ada Applications and Environments, Miami Beach, Florida, Apr. 1986.

[4] K. Kishida, L. Williams et al. SDA: A Novel Approach to Software Environment Design and Construction, to appear in Proceedings of the Tenth International Conference on Software Engineering, Singapore, Apr. 1988.

[5] P. Freeman, A Conceptual Analysis of the Draco Approach to Constructing Software Systems, IEEE Transactions on Software Engineering, 1987.

[6] Niklaus Wirth, PROGRAMMING IN MODULA-2 3rd Corrected Edition, Springer-Verlag, 1985.