# 実行可能な仕様書：
# データモデルに駆動される実装合理化技術の仕組み
## *Executable Specifications* Driven by Datamodel

渡辺幸三
WATANABE Kozo

有限会社ディービーコンセプト代表
DB Concept Co. Ltd. CEO

**要旨**

本研究では、情報システム開発の分野における「実行可能な仕様書」の動向や仕組みについて論ずる。この技術が実用化されたのが他の分野に比べて遅れたのは、情報システムが複雑な構造を持つデータベースを基礎としていたためと考えられる。情報システムが置かれた社会的文脈に合わせて、そのデータベース構造は的確に設計されなければならない。それは簡単な課題ではなく、データモデリングと呼ばれる専門的アプローチが欠かせない。この課題を克服すれば、「実行可能な仕様書」はシステム開発において強力な合理化手段となり、業界のあり方まで変革するだろう。

## 1. はじめに

筆者は民間企業や行政・自治体向け情報システムの開発に 30 年以上携わってきた現役技術者であるが、最初に就職した開発会社では、ワープロや MS-Excel を使って「仕様書（プログラム仕様書）」を書いていた。この資料にもとづいて別要員にプログラミングしてもらうためだ。この伝統的なやり方は今でも多くの開発会社が踏襲しているが、さまざまな矛盾や非効率をはらんでいる。

たとえば、仕様書を手間暇かけてこまごまと書けば、プログラミングと同程度かそれ以上の手間がかかって馬鹿馬鹿しくなる。しかも、コードのあり方と仕様書の内容が厳密に対応するため、それらの整合性維持に大変な手間がかかる。これを避けるために内容を概要だけにとどめておこうとすれば、プログラマによってコーディングされる結果に違いが生じ過ぎるために品質統制できないし、そもそも仕様書の存在意義が失われる。適切な詳細さで仕様書の内容を維持すればよいと言うのは簡単だが、それをやるには想像以上のコストがかかる。

こういった矛盾の本質は「書かれる仕様書の形式」にある。すなわち、書いた仕様書が「そのままアプリとして実行できる形式」を備えていれば、これらの問題は生じない。じつはそのための技術、すなわち「実行可能な仕様書」はすでに実現されている。実現されているからには、「そのまま実行できない仕様書」を手間暇かけて作成することに、もはや経済合理性はない。

## 2. 「実行可能な仕様書」の動作原理

「実行可能な仕様書」の動作原理を説明しよう。後述するように、この技術はさまざまなドメイン（ソフトウエアの適用分野）で実用化されているが、基本的な枠組みは同じである。

その基礎は「仕様書（定義情報）の形式」にある。まず、書かれた仕様書の内容がコンピュータ（シーケンサー）に理解可能、かつ実行可能なものでなければならない。ただし、そのままでは人間にとっては理解しにくいので、専用のエディタ（仕様書エディタ）が必要になる。これで仕様書データを開けば、「人間（開発者）にとってわかりやすい仕様書」に見えるし、そのまま編集もできる（図１）。

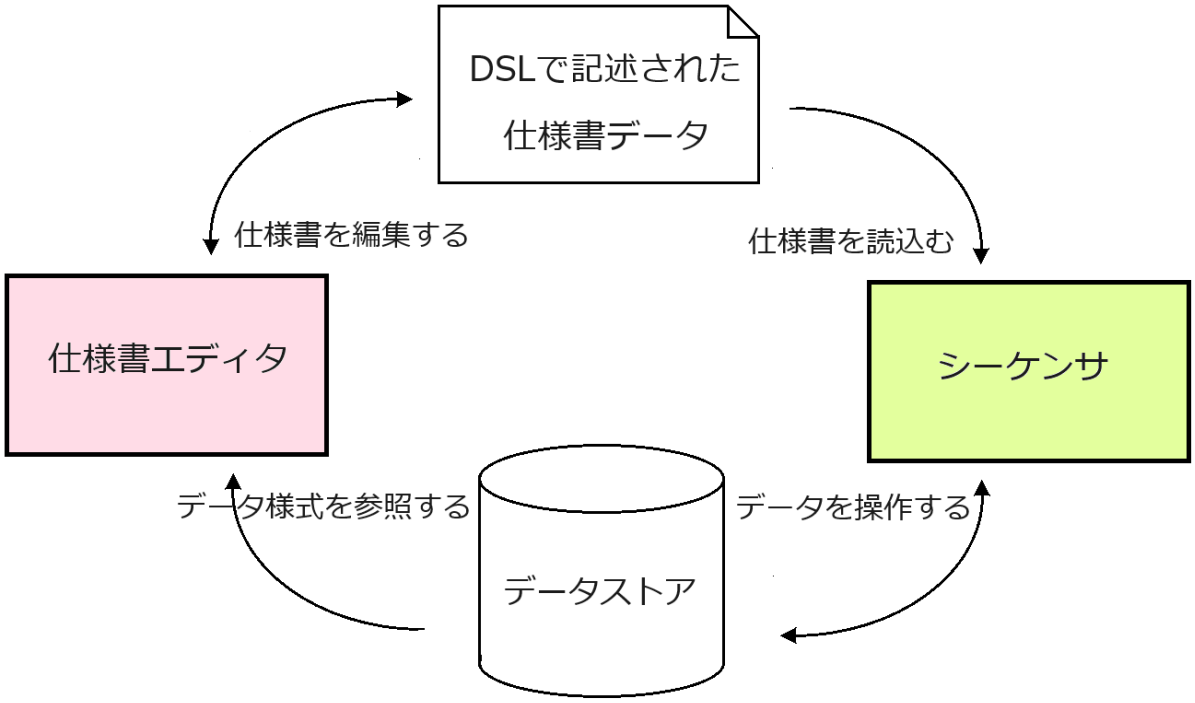


図１．実行可能な仕様書の仕組み

仕様書に付与される形式はドメイン毎に異なるため、一般に DSL（ドメイン特化言語、Domain Specific Language）と呼ばれている。この仕組み全体を、DSL の発展形の意味合いで DSP（ドメイン特化基盤、Domain Specific Platform）と呼びたい。

厳密に言えば DSP のシーケンサーは「動的制御方式」と「コード生成方式」に分類できる。前者では仕様書の内容が読み込まれると同時に実行されるし、後者では汎用言語のプログラムコードに変換・コンパイルされたうえで実行される。それぞれに特性や利点があるが、方式の違いそのものは開発者にとっては透過的なので本稿では問題にしない。

DSP に関わる４つ目の要素として、「シーケンサーの実行過程で動的に操作されるデータとその様式」が存在する。操作されるデータの源泉は、センサー等の機械装置、あるいはデータベース等のデータストア等さまざまである。シーケンサーがそれらのデータを扱えるように、そのデータ様式を参照する形で、仕様書の内容が編集される必要がある。後述するが、それらのデータの様式や操作の複雑さによって、DSP の実用化難易度が違ってくる。

## 3. 他分野での「実行可能な仕様書」の例

MS-Excel や MS-Word といったビジネスアプリも DSL を基礎とする DSP の一種とみなせそうだが、それらには「シーケンサー」の要素が欠けているので純然たる DSP とはいえない。DSP により近いものを挙げるならば Power Point で、専用エディタで編集された「スライドの仕様書」をシーケンサーが読み込めば、「仕様書どおりのスライドショー」が実行される。より典型的な DSP の例を３つの分野毎に説明する。

### 3.1. デスクトップミュージックにおける DSP

パーソナルコンピュータ上で音楽を演奏させる DTM（デスクトップミュージック）が、おそらく世界で初めて DSP を実現した分野である。基礎となる DSL は、MIDI(Musical Instrument Digital Interface)と呼

ばれる国際規格で、ヤマハやローランドといった国内外の楽器メーカーの連携によって最初のバージョンが生み出されたのは 1981 年のことだ。その後に急速に発展した DSP は DAW（Digital Audio Workbench）と呼ばれ、今では音楽業界にとって欠かせない技術として活用されている。

MIDI によって、楽曲のタイトルや調性、含まれる音符のタイミングや長さ、さらに音色やイントネーションまでが形式的に記述される。これをシーケンサーに読み込めば、指示にしたがった演奏が再現される。

いっぽうこの定義ファイルを専用エディタで読み込めば、一般的な楽譜、あるいはピアノロール（ピアノの自動演奏装置で古くから利用されてきた形式に沿った譜面）、あるいはタブ譜（ギターの指盤上の押さえ方とタイミングを示す譜面）といったさまざまな形式で表示される。

これらは見かけ上、「その楽曲の演奏仕様書」にしか見えないうえ、それぞれの形式のままで編集も出来るため、開発者（楽曲の編集者）にとっての便宜ははかりしれない。なによりも MIDI のこまごました体系やルールの知識が不要で、エディタの使い方さえ理解すれば利用できる点が重要である。MIDI は DSL として決定的な要素でありながら、開発者には周到に隠蔽されているということだ。

音楽業界での DSP の実現が世界初と見られるのは、音楽演奏という営為がもともと機械演奏に馴染みやすいものであったためであろう。また、音楽的表現を MIDI で記述するための規則は複雑かつ多岐にわたるが、演奏の最中に何らかのデータを参照する必要がない（厳密には演奏の開始や中止等の指示データのみを参照する）という意味で、DSP の構成は比較的単純で済む。

### 3.2. ロボット制御における DSP

機械の動作を制御する「制御系ソフトウエア」の業界において、DSP は 90 年代に実用化され「モデルベース開発」と呼ばれている。とくに工場で活躍する工作用ロボットには、使いやすい DSP が欠かせない。工程の諸条件に合わせてロボットの「動作仕様書」を専用エディタで書けばその通りに動作するし、動作を変えたければ仕様書を修正すればよい。

DTM に 10 年遅れて DSP が実現された理由として、ロボット制御においては「操作対象の状態」に応じて動作を動的に変化させる必要があったためと考えられる。つまり、操作対象の状態はセンサーによって認識されるが、認識結果に応じて動作が変化しなければいけない。単純な工程では不要ではあるが、複雑な工程ではセンシングによるリアルタイムの情報収集が必要になる。

したがって、ロボットの動作を仕様書として記述する際にも、操作対象の状態を表すデータの形式に沿わなければならない。これが DTM と異なる点で、ロボット制御において DSP 実現が遅れた主要な理由とみなせるだろう。

### 3.3. 情報システムにおける DSP

情報システム開発において DSP が実用化されたのは 2000 年代に入ってからで、日本では「ローコード開発ツール」と呼ばれるようになった。ロボット制御よりも実現がさらに 10 年遅れたのは、アプリの動作中に参照されるデータの構造が複雑、かつ、データの更新操作までをともなうためだったと考えられる。

情報システムが操作（読取および更新）するデータは、一般的に RDB（リレーショナルデータベース）に格納されており、その様式はシステム毎に異なる。ロボット制御においてセンサーから送られてくるデータの様式は固定的であるうえ、データが読み取られるいっぽうで更新されることがないのと対照的だ。

データ様式がシステム毎に異なるというのは、システム毎のデータ様式の設定が自由ということでもある。この自由度の高さゆえに、データの不整合や矛盾を許すようなデータベース構造が設計されがちだ。そのような設計段階で失敗しているデータベースを扱う限り、どんなに強力な DSP を用いたとしてもパワーは生かせない。

データベース設計の複雑さは、第 1 正規形、第 2 正規形、第 3 正規形、BC 正規形といった煩雑な専

門用語で知られている。ようするにこれらは、データ項目の定義域制約、およびデータ項目間の関数従属性を構成するためのルールである。

具体的には、100 のオーダのテーブルで構成されるデータベースにおいて、ひとつの事実を 1 か所に記録する（one fact in one place）ためのルールである。ひとつの事実が複数の場所で記録されることを許せば、遅かれ早かれアノマリー（更新時異状。データの不整合や矛盾のこと）が生じる。アノマリーの発生を避けつつデータベースを適切に設計する作業を「データモデリング」といい、その成果物が「データモデル」である。

DSP によってアプリの組み立ては依然と比べて格段に容易になったが、扱われるデータベースにアノマリーが含まれている限り、経営に資する情報システムは手に入らない。実際のところ、適切にデータモデリングされているシステム事例はきわめて少ない（筆者の印象では 1%以下）。したがって、DSP を生かせる案件は、システムの全面刷新や新事業のためのスクラッチ開発といった、データベース構造の見直しをともなうものに限定されるだろう。

DSP のもうひとつの課題を挙げておこう。MIDI が国際規格であると説明したが、ロボット制御ではもう少し緩いデファクトスタンダードがある。しかし、情報システム開発における DSL には国際規格もデファクトスタンダードもない。それゆえに、DSP 間で定義データを再利用することが難しい。データベース定義については SQL として国際規格が実現されているが、アプリ定義（本稿で言う仕様書）については形式がまったく統一されていないのが実情である。

にもかかわらず、DSP の導入には大きな意義がある。2000 年代以降、情報システムは RDB を核としながらもオブジェクト指向言語で開発されることが多かった。この組み合わせの場合、ORM（オブジェクト・リレーショナル・マッピング）と呼ばれるコストパフォーマンスの悪いやり方をとらざるを得ない。出来上がるシステムの保守性が悪いし、データベースが「ただのオブジェクト（クラスインスタンス）の永続先」とみなされるため、本来の RDB の強みを生かせない。DSP の導入によってオブジェクト指向開発が不要になるだけでも、そういった混乱が緩和される可能性がある。

筆者はオブジェクト指向開発を否定しているわけではない。個々の案件をオブジェクト指向言語でいちいち手作りするのはあまりに手間がかかる。しかし、DSP のエディタやシーケンサーといった要素をオブジェクト指向開発して活用すれば、本来のパワーを発揮できると考えている。じっさい筆者は自作の DSP を Java で個人開発した（延べ工期 2 年、6 万ステップのプロジェクトになった）のだが、今ではそのツールを用いて桁違いの生産性を実感できている。

## 4. 「実行可能な仕様書」が業界にもたらす影響

情報システム開発の業界は生産性が低く労働集約的と言われてきたが、DSP はそういったあり方を変革する可能性がある。DSP 導入にともなって顕著に変わるのが、チームのサイズや工数だ。

なにしろ、「設計専任要員」と「プログラミング専任要員」との分業が要らない。設計を担える要員が仕様書を書くだけで実装とユニットテストが完了するので、詳細設計からプログラミング／ユニットテストまでの工数や工期が大幅に削減される。設計担当者１名に対してプログラマは２～５名あてがわれるので、チームのサイズと工数は３分の１以下、工期は半分以下になるだろう。

また、Java のような汎用言語を用いた開発スタイルは、いわゆる「バックエンド」と「フロントエンド」の大げさな分業をもたらしたが、これも DSP の導入で無用になる。DSP によってバックエンドとフロントエンドが統合されているからだ。

チームが少数化するだけではなく「精鋭」をチームにあてがう必要がある。上述したように、DSP を有効利用するためには、的確なデータモデルが確立されなければいけない。多くの開発企業ではデータモデリングのトレーニングがまともになされていないのが現状で、100 個のテーブルを含むデータベースの設計を的確にこなせる技術者は払底している。払底していると言うよりは「いない」と言ったほうがいい。

多くのシステム開発企業は「工数の大幅削減」にも「精鋭技術者のプロジェクトへの提供」にも対応できそうにない。彼らのビジネスモデルが「莫大な工数を見積もって、大量の人員を外注企業から派遣する」というスタイルで固定化しているからだ。彼らの売上げは、多人数の外注要員を管理するためのプロジェクト管理（という名の外注管理）の費用と、外注からのマージンで構成されている。そのようなスタイルで 30 年以上続けてきたので、軌道修正は簡単ではない。

彼らにとって DSP を導入しにくいもうひとつの理由が、DSP が彼らにとって不得意な「スクラッチ開発」を想定している点にある。

2000 年代に ERP パッケージシステムを導入してカスタマイズする手法が国内外で流行した。「グローバルスタンダード」の謳い文句につられて多くの大企業がこぞって導入した結果、大手のシステム開発会社はそのトレンドに沿って人員を最適化した。すなわち、いわゆる「フィット&ギャップ分析」を実施してアドオン（パッケージでカバーできない機能のこと）をまとめる作業をもっぱらとする技術者が大量に育成された。顧客の意向を聞き出してゼロからシステム仕様を構想できる技術者は、今では例外的といっていいほど少ない。

いずれにせよシステム開発業界のあり方は、DSP の登場・普及で少なからず揺らぐだろう。的確にデータモデリングして DSP を活用する辣腕技術者は、少数ではあるが確実に社会貢献できる。いっぽう、従来のやり方を墨守するばかりの大手 SIer は今後も ERP パッケージ導入やオブジェクト指向開発にこだわりそうだ。そういった業者間のパフォーマンスの二極化を知れば、遅かれ早かれ顧客も大手 SIer のやり方に疑問を持つだろう。そのとき、大手 SIer はビジネスモデルを果敢に変えて状況に対応するかもしれないし、変化できずに縮小していくかもしれない。

## 5. まとめ

ソフトウエアはさまざまなドメインで利用されている。そのドメインに経済的活力があれば、そのドメインで使える DSP は遅かれ早かれ生み出される。なぜなら、優れた DSP を生み出して協業他社を出し抜こうと考える業者が、自由競争市場において必ず現れるからだ。

また DSP は、それぞれのドメインに新たな魅力を与える役割も持っている。開発生産性を桁違いに高める DSP を使いこなすためには、そのドメインに関する深い知見と高度専門職としての適性が厳しく求められる。チームは少数精鋭化するし、同時に IT 技術者の待遇が改善される。結果的に、有能な若手がそのドメインに流れ込み、技術革新がさらに進展することが期待できる。

いずれにせよ DSP の登場は歴史的必然であり、これがシステム開発業界に古くからある問題を刷新してくれることを筆者は望んでいる。そのためにも、データベースを適切に設計するための基本技術の普及が欠かせないと考えている。