業務アプリケーション開発用途向け What 記述指向言語の開発
—業務ロジック記述言語 SPECRIPT—

1. 背景

業務アプリケーションは、企業活動の変化に伴って仕様変更・拡張を繰り返すメンテナンスフェーズがシステムライフサイクルのほとんどを占める。提案者はこの 10 数年、その業務アプリケーション開発に携わってきた。その中で眼にしてきた多くの業務システム開発の現場は、

- ◆ 年々強まる短工期のプレッシャー
- ◆ 多彩な商品開発に伴う頻繁な仕様変更要求
- ◆ ドキュメントとソースコードの同期のためのコスト増、そして乖離
- ◆ 人材の入れ替わりによる仕様把握の低下
- ◆ 仕様変更対応における、変更箇所の影響範囲調査ミスによるトラブル

といった要因の悪循環によりエンジニアの負担が大きく、特にメンテナンスフェーズにおいては危機的状況にある。

これまでは基盤構築や開発規約の整備、フレームワークの導入、そしてオフショア利用等による開発コスト抑制を行っていくことで何とか凌いでいる所であると言えるが、より短期間での開発が求められる一方、担当者や開発者が入れ替わる際に十分な時間をもって引き継ぎが行われる事は珍しく、ドキュメントに記載されなかった仕様に関する暗黙知の情報は少しずつ欠落していき、結果として潜在的なトラブル要因が増えていくといった事が日常茶飯事である。つまるところ業務システム開発が属人的である事から脱却できずにいるのが現実で、このままでは何か抜本的なブレークスルーがない限りエンジニアへの負担が増え続ける事は容易に想像できるであろう。

2. 目的

そこで我々は、システムの仕様を容易かつ的確に把握できるようにプログラムコードに仕様を直接記述できる新言語を作成し、その実行環境を含め提供できれば、仕様の管理・把握、引継ぎの負担が大幅に軽減され、そのブレークスルーとして大きな一歩になるのではないかという考えに至った。これまでの開発言語では業務仕様を記述するには低級すぎるため、処理の制御構造のための記述が多く入り込み、仕様を読み取りにくくする大きな障害となっていた。

またプログラムジェネレータ的アプローチは今も昔もそこかしこで行われているが、技術的に困難なためか、なかなか決定版もなく、少し使えそうなものがあっても高価で導入が容易でなかったりするのが現状である。しかし「青いバラ」を待っている余裕はもはや現場にはないのである。

この新言語開発プランは、更にドキュメント生成機能を派生させることで、ドキュメントとコードの同期のための工数や乖離を排除することが可能となる。この新言語が完成した暁には、仕様は全てプログラムコード上に記述されており、その構文解析も完了しているので、比較的容易にドキュメント生成機能が実現可能になるからである。

本プロジェクトの目的は、工数やスケジュールの関係から、このブレークスルーための布石となる新言語をまず作成することとする。将来的にこの新言語をコアとして現在の問題点に関しての様々なブレークスルーを発展させることができると考えている。尚、この新言語は汎用的に何でも記述できる事を目指すものではない。例えばリアルタイムによるハードウェア制御等は対象とはしない。記述できる業務として想定しているのは、「販売管理」や「会計管理」「営業管理」等に代表される企業活動のシステム化における処理ロジックを扱うものである。

3. 開発の内容

本プロジェクトの目標は、業務アプリケーション開発用途に限定し、仕様記述といえるレベルの記述性を備えた新言語「SPECRIPT」を開発することにある。

SPECRIPT は、業務アプリケーションのビジネスロジック記述に特化した言語である。業務アプリケーションのアーキテクチャーとして、プレゼンテーション⇔ビジネスロジック⇔データアクセスの3層モデルをとった場合において、中間層であるビジネスロジック層の記述を担うことになる。しかし、この言語は最近の流れのようにオブジェクト指向を目指した言語ではない。我々は長年の業務アプリケーション開発の経験から、一般にビジネスロジックはオブジェクトモデルで表現するよりも、業務処理毎の視点から求められる「①データの有り様(必要な属性や値の制約)」、その有り様に準拠している事を前提に行われる「②データ処理」、実際にシステムで扱う「③データそのもの」という切り口で表現した方が、以下の

- ◆ この業務処理で行っている事は何なのか？
- ◆ 入力として期待されているのはどのような構造を持ったデータなのか？
- ◆ 処理対象のデータ値には、業務上どのような制約があるのか？
- ◆ この業務処理の結果、どのような構造のデータが出力されるのか？

について明確に記述でき、かつ効率的に管理が行えるはずであるという考えに至った。

そこで SPECRIPT では、そのデータの有り様を spec、一連のデータ処理を function、また、データについては property という言語要素で表現することにし、一つの業務処理に関連する各要素を一つのファイルに宣言的に記述していく事とした。

SPECRIPT には言語仕様として以下のような特徴を持たせる。

- ◆ 手続き的な記述(＝'How'的な記述)を極力排除し、極限まで宣言的(＝'What'的)に記述できるようにする。→ What 指向
- ◆ 業務処理手順を表現する場合にどうしても残る手続き的記述に対しては、以下のような文法上の工夫を行い、記述したコードがそのまま形式的な'機能仕様記述'であると言うことができるようなレベルの記述性を確保する。
- ◆ 繰り返し処理を抽象化した特別な"cumulate"関数や list に対する集合処理関数(select, convert, satisfyAll, satisfyAny, ...)を導入し、ループ構造を排除する。
- ◆ 拡張多分岐演算子を導入して、ネストした条件分岐構造を排除する。
- ◆ 代入処理(変数)の廃止を排除する。
- ◆ ネームスペースを導入し、記述する業務仕様や処理をカテゴライズして開発者が管理・認識しやすいようにする。
- ◆ 実行環境は用意するが、リソース接続情報に関する諸事項・デプロイ環境に関する諸事項は、実行環境に対するコンフィギュレーションとし、言語仕様の構成要素からは切り離す。

SPECRIPT 実行環境は、そのコアは Scheme で、実行環境全体は Java で実装され、JVM 上で外部 Java プログラムからコールされる形で協調動作する事になる。コンパイルされた SPECRIPT コード(実体は S 式:Scheme コード)は、この Java＋Scheme で実装された SPECIRPT 実行環境上でインタプリットされ、実行される。

例えば Web アプリケーションであれば、プレゼンテーション層で行うべきページ遷移の制御やユーザー入力の受付け、結果表示処理は Tomcat、Struts 等で、その部分だけを記述する。ビジネスロジック層ではプレゼンテーション層で受けた入力を基に実際に検索を行ったり、業務処理を行う部分を SPECRIPT で記述する。プレゼンテーション層の GUI からのアクションによりバインドされたビジネスロジック層の SPECRIPT コード(業務処理)がコールされ処理が実行される。

入力値のチェックすらも業務の要請する制約としてビジネスロジック層に記述されるので、プレゼンテーション層で実装する必要はない。ビジネスロジック層での処理で必要があれば、SPECRIPT が提供するライブラリを通してデータアクセス層である外部リソース（ファイルやデータベース等）とやり取りを行う、といった形で実現される。

SPECRIPT は、SPECRIPT Compiler、SPECRIPT Runtime、ResourceManager の3つの部分で構成される。

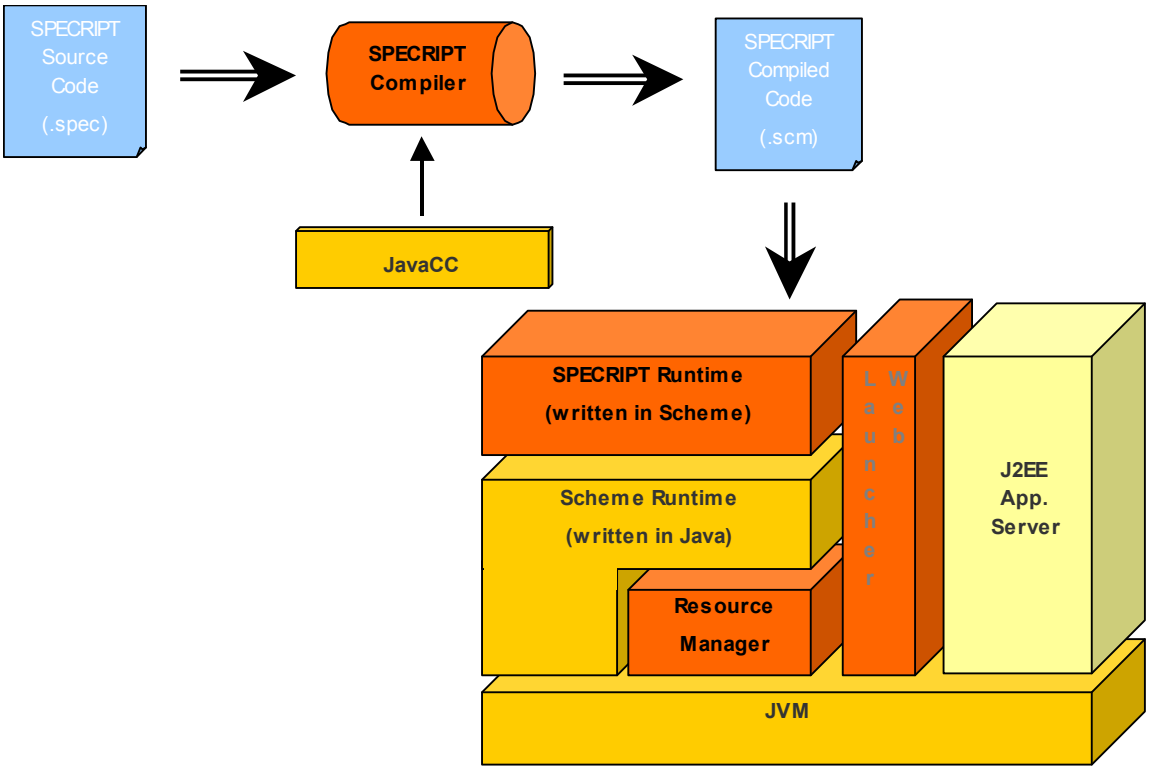

図1：ソフトウェア構成図

4. 従来の技術（または機能）との相違

◆ MDA との関係

MDA の発想と‘超高級言語’としての SPECRIPT の発想は、ほとんど同じものであると理解している。MDA におけるモデル記述と SPECRIPT による仕様記述は、ほぼ同義であると認識される。この理解の元では、SPECRIPT の実装は MDA を実現する実装のひとつである、と捉えられることとなる。

一方、次のようなアプローチに対する違いがある。MDA が、UML クラス図による静的構造記述とそれにぶら下がるかたちのアクションセマンティクス等による動的振る舞い記述、というアプローチであるならば、SPECRIPT では振る舞いが function というかたちで公開されており、静的構造は spec や property 定義などで示され実行時に検査されるのみである。言い換えると、MDA がオブジェクト指向的であるのに対し、SPECRIPT は‘脱オブジェクト指向的’と言える。

◆ BPEL との関係

BPEL は、業務フロー記述に特化しているが、そのフローで駆動される Web Service として提供されるような個々の業務ロジックは、何らかの実装言語で実装（現実には Java で実装）しなければならない。BPEL から個々の業務ロジックの実装の中身を生成することは出来ない。SPECRIPT は個々の業務ロジックの中身を記述する言語であると位置づけられるため、BPEL とは共棲関係にあると言える。

◆ Java・C#との違い

SPECRIPT では記述する範囲をプレゼンテーション層やデータアクセス層から分離した「ビジネスロジック層に限定」することによって、アプリケーションの実装方法による影響を受けることが無くなり、設計次第では緩く成りがちな基盤コードと業務コードの分離が強制され、真の業務ロジックの再利用性が高める事が可能である。

また、高度に抽象化された記述性と言語機能により、制御構造等もコード上には現れず、もはや実装記述というよりも、詳細設計記述と言える。

5. 期待される効果

SPECRIPT はタイトルにもあるように業務アプリケーション開発にターゲットを絞っている。しかも業務ロジックの記述に特化した言語である。以下の特徴を持たせることで、より業務仕様の把握やメンテナンスにおけるバグの混入を防ぐことが可能となる。

◆ ソースコードからの仕様可読性が高い

制約や not null などの厳密なデータ定義、処理のまとまりには function として名前付けをしなければならないこと、これらがあいあまって、ソースコードからの仕様の可読性が高いと言える。

◆ 業務ルールをデータに関連付けつつ取り纏めることができる

- spec 定義に、そのデータの取り扱いに関わる業務ルールを制約として記述することができる。
- 業務ルールに違反するデータがあれば確実に検知される。
- データの処理手順(※C/R/U/D など)から業務ルールを確実に分離できる。

◆ 想定外のデータの発生を確実に検知できる

- spec 定義に、値内容に踏み込んだデータ項目の規定(＝仕様)を盛り込むことができる。
- 制約違反があれば確実に検知される。

◆ ケース漏れがおきない(※少なくとも起きにくい)

- 条件演算子 "?" ～ ":" ～ の場合、「else」の記述が必須。
- 多分岐演算子"==?"や"in?"の場合、「default」の記述が不可。

◆ 業務の依存関係(あるいは業務のコンテキスト)を明示できる

- 業務構造を表すように namespace 階層を構築することにより、業務の依存関係、もしくは業務の‘文脈’がソースコード上で明示的となる。
- 複数の依存関係があった場合も、複数の namespace を付与することで対応できる。

6. 普及(または活用)の見通し

業務システムを開発している同じ問題意識を持つ人達に活用していただく事を目標に、オープンソースとして公開していく。またコミュニティも設け、業務システム開発の現状をより良くしたいという有志を募り、更なるブラッシュアップをしていきたいと考えている。

7. 開発者名(所属)

田中　浩一　(有限会社　マルチパラダイムシステムズ)
長村　州浩　(有限会社　マルチパラダイムシステムズ)
佐藤　賢司　(有限会社　マルチパラダイムシステムズ)

(参考)http://www.specript.org　(近日公開予定。言語仕様等の更なる詳細資料も掲載予定。)