Domain-Specific Modeling: How to Start Defining Your Own Language[1]

http://www.devx.com

あらゆるアプリケーションや産業分野、課題に適合するような抽象的な方法を避けること、いわゆるDomain-Specific Language により、開発における妥協を回避できる。 独自のDSLを構築することで、どの様にして、品質と生産性を大幅に向上できるかをご紹介します。

DSL（Domain Specific Language）は、もはや避けて通れないものになっています。UMLを多目的に用いることを止めて、DSLのサポートを表明するベンダーの数は増え続けています。しかし、それはなぜでしょうか？

DSLはより表現性に富んでおり、それゆえに複雑なものへの対処を容易にし、簡単にモデル化できる便利な手法です。さらに重要なことは、DSLを使えば、今日コンパイラが他のプログラム言語からアセンブラを生成するのと同じ方法で、自動的に完全なコード生成が行えることです。

DSM（Domain-specific modeling）は、モデルを中心に開発を進めることで、真のモデル駆動型開発を行える環境を提供します。また、ユーザー固有のDSLの定義と保守が出来ることが重要な点です。言語が真にドメイン固有（Domain-specific）であれば（言語が単一の企業または企業内の単一の開発に限定されているのが理想です）、モデル駆動型開発は最良の結果をもたらします。

設計した言語の文法中に妥協が全くなければ、その言語から生成されたコードにも妥協はありません。これがモデル駆動型開発の本質であり、もっとも生産性と品質を向上させる近道です。慎重に作りこまれ、詳細に定義されたDSLは、非常に大きな生産性と品質の向上をもたらします。

企業、正確には企業の開発エキスパートは、如何にして安定したDSM言語を作り出せばよいかという問題に直面するでしょう。新たなモデル化言語を定義することは、これまでのスキルにはないものであり、これは当然起こりうる問題です。ほとんどの開発者は、モデル化言語の抽象度を上げることで、もっとも大きな利益が得られることを知っています。しかしながら、初めてそのような言語を作成する仕事を与えられた時、往々にして普段書いているコードを反映したモデル化言語を作ってしまいます。このような言語では抽象度のレベルがあまり上がらず、ソースコードの自動生成による生産性の向上に歯止めをかけてしまいます。

この資料では、効果的なDSLを作成するための幾つかのガイドラインを提供するために、１５年間この領域での経験してきたことを記述します。単なるドキュメント生成を超えた、モデル作成者が完全な製品レベルコードを生成する事を目的としたモデル化言語を紹介します。

---

[1] 原文 http://www.devx.com/enterprise/Article/30550

モデル化言語の作成は非常に難しい仕事のように感じるかもしれません。汎用のモデル化言語を構築するならば、その通りです。もしモデルにしようとしているアプリケーションの形態が分からない場合に、どのようにしてモデリング言語を作成しますか？それは非常に困難な作業であり、多分、完全な製品コードを生成するためには貧弱であると良く知られているUMLのようなものになってしまうでしょう。

ある会社の単一ドメインに対象を限定すれば、言語の定義は相当簡単な作業になります。作業が簡単になることで、今現在作業をしているドメインに集中することが出来ます。言語の有効範囲を狭めることでも、設計の抽象度を上げることができ、自動開発の為のコードジェネレータ作成が簡単になります。

DSMを定義するためのステップ

モデル化言語作成は、言語を徐々に成長させながら進めるのが最良の方法だと解ってきました。言語を少し作り、モデルを少し作り、コードジェネレータを定義する。言語に幾つかの変更を加え、モデルを幾つか追加し、、、、といった具合です。

モデルベースの開発プロセスは、以下の４つのフェーズに分解できます。

１．　抽象化したもの、及びそれらが各々相互にどう働くかを見つける
２．　言語のコンセプトとルールを規定する。（METAMODEL）
３．　言語の視覚的な表示を作成する（NOTATION）
４．　モデルのチェック、コード生成、ドキュメント生成等のためにジェネレータを定義する

通常、プロセスもこの順番で始まります。ここから、各ステップを少し詳細に見て行きます。

１．抽象化：

モデルをコードの視覚化と捉えてはいけません。どの様なDSLに於いても、抽象化は成功のための最重要点です。正しい抽象化の方法を明確に見つけることが、最も重要な点です。（クラス図をクラス構造に結びつけるような）低レベルの言語コンセプトがもっとも簡単な方策であるように見えるかもしれませんが、コンセプトを対象とするドメインの課題に割り当てる（map）ほうがより良い方策であり、それによって抽象化のレベルが上がります。この方策を採る事で、設計フェーズの早い段階でエラーが起こらないようにでき、仕様化作業を最小にでき、同時にコードの自動生成により適した言語を作成できます。実装のためのコンセプトではなく、対象ドメインごとの用語で記述すれば、将来の検証にも役立ちます。言い換えれば、アプリケーションが何をするか(what)が重要なのであり、どう実現されるか(how)や使用する言語、或いはフレームワークの種類が重要なのではありません。

２．言語の構造：

開発者には、抽象化物の理解、アーキテクチャ上のルールの踏襲、モデル部品の適切な再利用等が求められます。抽象化物とルールを正しく定義した言語を使用することで、開発者は（おそらく旧態依然とした）内製の”デザインガイドライン”を常に参照しなければならないと言う事態から開放されます。

モデル化のコンセプト、属性及びルールがこのフェーズで規定され、モデルが正しく作成されるように制御できるようになります。その後、他のコンセプトがオブジェクト属性、接続、サブモデル或いは他の言語へのリンクとして取り込まれていく間に、大きなドメインコンセプトがモデル化言語に結び付いてゆきます。

３．表示方法：

次に、言語の視覚的な表現が必要です。通常ダイアグラムで表されますが、マトリクステーブルや単にテキストで表されることもあります。この資料の図は、シンボルやアイコンが異なった言語コンセプトを表す、幾つかのグラフィカルなモデル化言語を例示しています。優れたDSMツールを使えば、モデルを簡単に作成し、読み取り、保守できるような独自の表示方法を定義できます。UMLの様に、全ての異なったコンセプトに四辺形を使うことは、文字が A だけで、ほんの僅かなその変化形しかないような外国語を理解しようとするようなものです。

４．ジェネレータ（ソースコードジェネレータ）：

モデルを、実行ファイルにコンパイル可能な、或いはインタプリタで実行可能なソースコードに変換することが究極の目的です。各々のモデルコンセプトをソースコードやその他の出力ドキュメントに結びつける方法を、ジェネレータを作成することで定義します。もっとも単純なケースは、モデル化された各々のシンボルに設定された値を、テンプレートの引数として使用したソースコードを生成することです。一般的にジェネレータは、他のシンボルとのリレーションシップや他のモデルの情報も利用してコードを生成します。

DSM言語では、モデル化するエディタを提供するだけではなく、言語やジェネレータの定義を行え、開発者が更新した言語の新しいバージョンを共有でき、言語バージョンの更新が既存モデルに反映されるようなツールが求められます。幸いにして、このようなツールは既にあり、言語作成の容易さに応じて市販のツールから、研究用のツールから派生したコードフレームワークのようなものまで色々なツールがあります。これらのツールのリストは、http://www.dsmforum.org/tools.html にあります。このような機能を既に実装しているツールを使うことで、熟練の開発者は特定のドメインに特化したジェネレータや設計言語を定義することが出来ます。他の開発者は、結果として得られるDSM言語やツールを使って設計が行え、それらのモデルから、実行可能な製品コードを生成することが出来ます。

この資料は、DSM言語の作成を中心にかかれています。ジェネレータは、もちろん言語に深く結び付いていて、その定義は非常に面白いトピックですが、ここでは詳細には触れず、機会を改めて紹介いたします。

言語のコンセプトと抽象化物を見つける

DSLを定義することによる最終目的は、モデル作成者やプログラマーにシステムを構築できるような高レベルの言語を提供することです。言語のコンセプトを定義する場合に、狭義に限定されたアプリケーションドメインに注目することが非常に重要なことで、要求の変化に応じて言語を変更できることも必

要です。この言語の更新への柔軟性が、DSMツールを選ぶ上での本質的な選択要素になります。良いツールは、このような変更や更新が可能で、以前に作られたモデルを言語の変更に応じて全て自動的にアップデートします。それに対して、完成度の低いツールでは、言語を変更すると全てのモデルが無くなってしまいます。

DSM言語を作成する人には、注目しているドメインで以前に幾つかの同じような製品を開発した経験のある開発者或いは、それらの製品のフレームワークやコンポーネントライブラリーの生成を行えるような開発者を当てることを勧めます。こういった開発者は、そのドメインにより詳しくて、それゆえ簡単にモデルのコンセプトを定義したり、ルールを規定したり出来ます。

全てのドメインは他のドメインと異なっているので､言語間でのコンセプトと抽象化物も異なってしまいます。コンポーネントサービス、既存のシステム定義、システムのアーキテクチャとドメインで使用されている用語を基に、最適な言語コンセプトを見出す事が出来ます。言い換えれば、組織内で使われているドメイン固有の専門用語や語彙を採り入れる事ができるでしょう。このような語彙を用いることで、既に理解している方法で生産物を定義する自然なコンセプトが得られます。（誰もコード生成の為の解決策だとは思わないでしょう。）既存の語彙を使うことで、新しい、馴染みのない用語の説明や既存の用語との対応付けをする必要がなくなります。

多くの場合、ドメイン上のある観点からDSLの生成を開始することになるでしょう。その後、言語のコンセプトが選択され、それらの定義が始まります。その基準となるのは、

・＜製品の物理的な構造＞
・＜システムのルックアンドフィール＞
・＜システムの拡張性＞
・＜ドメインエキスパートのコンセプト＞
・＜生成される出力ドキュメント(ソースコード)＞

ここから、それぞれのカテゴリーについて詳細に述べます。同時に幾つかの例も示します。

＜製品の物理的な構造＞

物理的な構造は、特定が容易で明確に定義できます。それゆえ、言語の定義を始める上で、良い開始点になります。発電所や紙工場のようなケースでは、バルブ、モーター、センサー、制御のようなコンセプトがあります。バルブは、”サイズ”と”方向”といった属性と、モーターやセンサーとどう結び付くかといったルールを持っています。DSM言語では、これと同様のコンセプトを言語構造として直接使用できるでしょう。

物理的構造をベースにした言語は、通常、静的な宣言を中心に定義されますが、振舞の要素が含まれる

こともあります。このような言語の設計では、後のソースコード生成プロセスに設定データを提供し、他のモデルとリンクされて、より包括的なソースコードを得ることが出来ます。

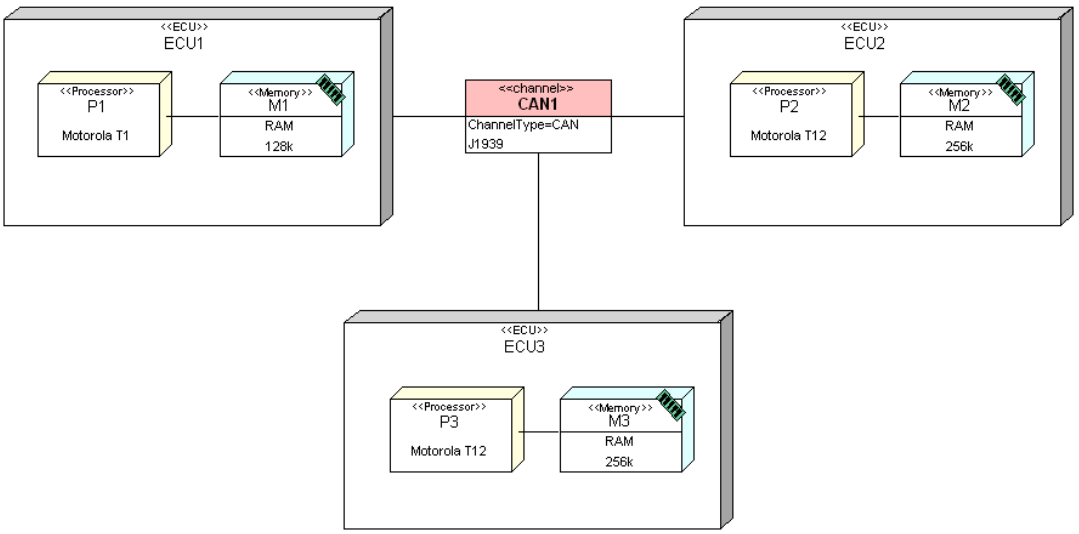

Figure1　物理的構造：この物理構造をベースにしたDSM言語は自動車のハードウェアアーキテクチャーのモデル化に使用されています。（EAST-ADLベース）

Figure1はEAST-ADLをベースにした物理的構造ベースのDSM言語の一部を示しており、DSLは自動車におけるハードウェアアーキテクチャーの定義の一部に着目しています。この図では、CPUが乗ったECUとメモリーがCANバスを通して繋がっているのが分かります。ここで定義されているハードウェアアーキテクチャー内の、様々なバスタイプとバスのアプリケーション上の制約がDSM言語として与えられます。

＜システムのルックアンドフィール＞

エンドユーザーの使用手順や製品用途を基本にした言語の定義も可能です。これを製品やシステムの”ルックアンドフィール”をベースにしたアプローチと呼びますが、あらゆる種類のシステム上の認知されている動作や遣り取りが、ここに含まれるでしょう。例えば音声応答メニューを定義した言語は”メニュー”、”プロンプト”、”音声入力”と言ったコンセプトを含んでおり、同様に、それらが実際のユーザーの使用手順とどう結びつくかのガイドラインも含んでいます。この種の言語は、それが実際の製品の視覚化された複製物であるために、非常に簡単に構築しテストすることが可能です。　課題は他のGUIを使用しないコンセプトへの対応付けです。

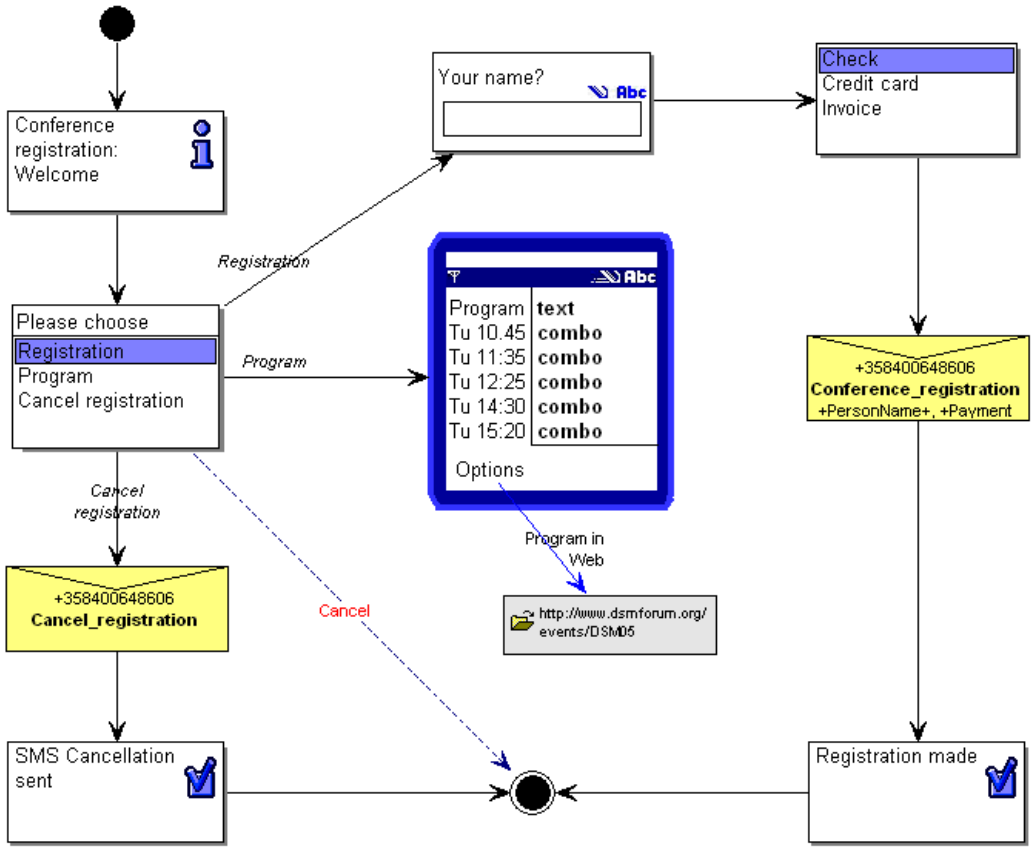

Figure2　ルックアンドフィール言語：S60　Symbian携帯電話とPDAをベースにしたルックアンドフィール言語

Figure2は、ルックアンドフィールをベースにした言語によるアプリケーション設計を示しています。この言語はSymbianベースの携帯電話やPDAをターゲットにしており、アプリケーションの振舞の定義が可能です。ルックアンドフィールは、携帯電話の実際のユーザインターフェイス部品を使用して表現されており、SMSの送出やWEBへの接続のようなプラットフォームが提供するサービスも表現しています。もし携帯電話でアドレス帳やカレンダーを使ったことがあれば、この単独のモデルを見ただけで、アプリケーションの動作が簡単に理解できるでしょう。

＜拡張性＞

言語の定義を始める別のアプローチとして拡張性への注目があります。拡張性のオプションを取り込むようなコンセプトを言語に定義することで、モデル作成者は複数の製品やその機能の差分だけに全力を注げます。

このタイプの言語定義では、将来の拡張に必要な余地の予測が成功の鍵です。拡張性を意図した言語は、”プロダクトライン開発”に最適です。言語定義の準備は、どの抽象化物が全製品に対して同じで、どれが違うかを規定するという意味で、徹底的なドメインの解析に繋がります。その後の言語は、どれが異なり得るかを記述する為にだけ使用されます。通常、開発者が数十の選択肢の為のパラメータ表やウィザードを作ることで、固定的な拡張に簡単に対処できます。

パラメータの選択が他のパラメータの選択に依存しているような場合には、事態はより複雑になります。実際には、機能とパラメータを選択するアプローチは、新たな機能や拡張性を作成する能力を要求したときに崩壊します。DSMでは、可能な限りの製品機能が決定される必要はないという状態をサポートすることで、解決策を提示します。

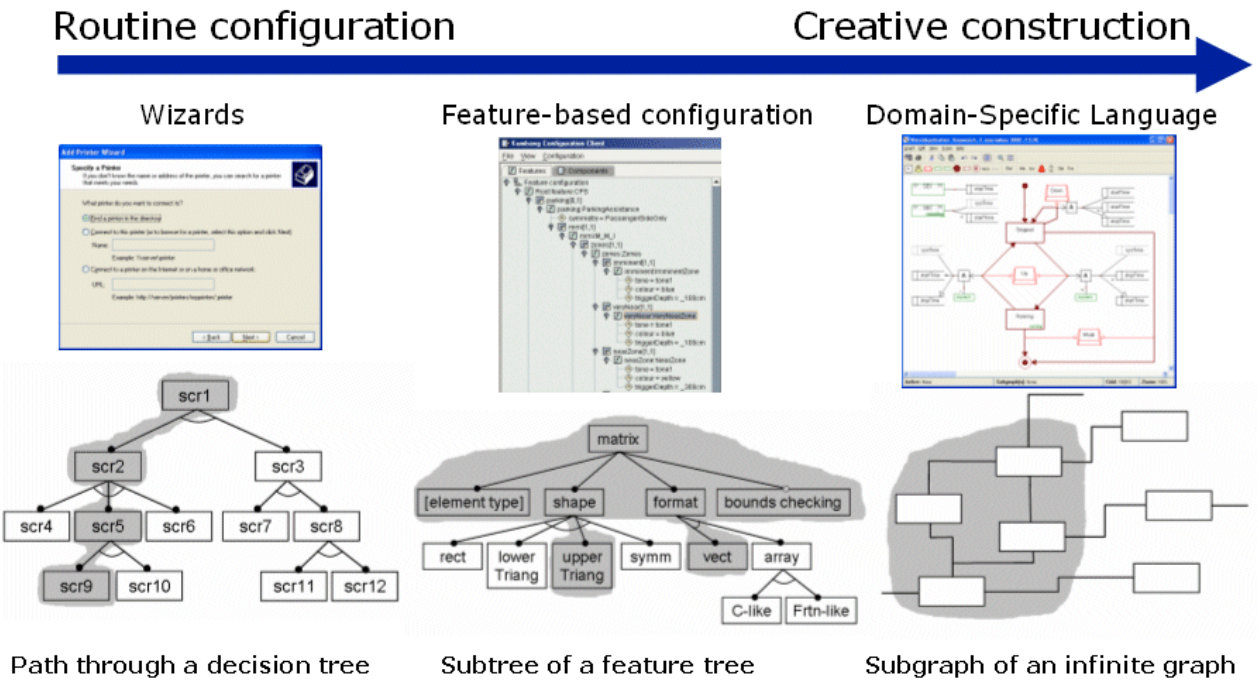

Figure3　拡張性の範囲：拡張性の範囲のメソッドが、言語のシンタックスによって包括されているような場合の判断を如何にして明らかにするかを示しています。(“Generative Programming, Methods, Tools, and Applications,” Czarnecki & Eisenecker, Addison-Weseley 2000.)

Figure3は拡張性の範囲を示しています。ウィザードや機能ベース（左から１，２番目の図）による構成の設定では既知の機能の中からしか選択が行われません。DSM言語では明白に選択の対象を設定しませんが、実質的には拡張を設定するための無限の空間を与えます。数え切れないほど多くなり得る、全ての拡張を知る必要はありません。より重要なことは、新たな機能をうまく追加出来るように言語を定義可能にできるということです。この言語を使えば、必然的に、モデル作成者が正当な機能と製品のみを作るように制限できます。

＜ドメインエキスパートのコンセプト＞

ドメインエキスパートがプログラマーではない場合は、プログラミングコンセプトとかけ離れたレベルまで抽象度を上げる必要性があります。ドメインエキスパートのコンセプトをベースにした言語は、相対的に簡単に定義できます。なぜなら、エキスパートにとって、ドメインは既に理解が確立したものだからです。ほとんどのモデル化のコンセプトは、ドメインモデルから直接導き出すことが出来るでしょう。幾つかの機能制限にも、同じことが当てはまります。

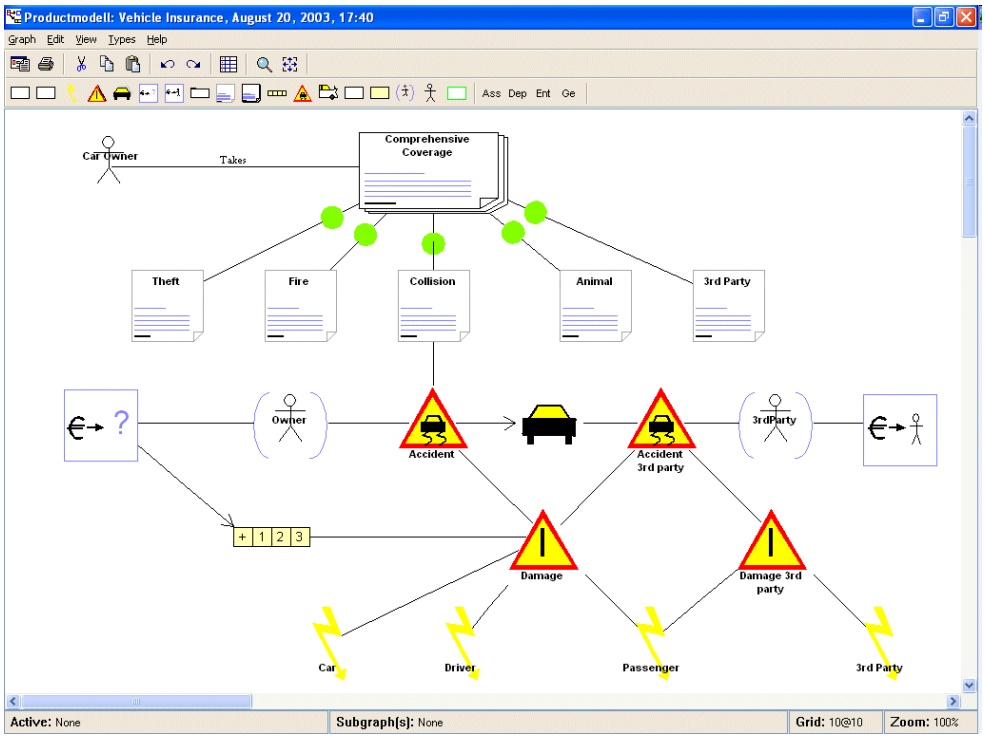

Figure4　ドメインエキスパートメソッド：ドメインエキスパートのコンセプトに基づいたDSM言語を使用したJ2EEウェブアプリケーションのための、金融・保険製品のモデル

Figure4はドメインエキスパートのコンセプトをベースにした言語を示しています。これは、モデル化のコンセプトを金融と保険商品に適応させた特殊な言語です。”リスク”、”ボーナス”、”ダメージ”といったコンセプトは、保険における当該の現実に対応しています。この言語を使うことで、プログラマーではない保険の専門家が、異なった保険商品を定義するためのモデルを書けます。ソースコードジェネレータは、彼らの設計をJ2EE　ウェブアプリケーションに変換します。この方法では、JAVAプログラムのエキスパートが言語とソースコードの関連付けを一度行えば、お互いに相手の領域の複雑さを知る必要がありません。ドメインエキスパートのコンセプトで使用している、より高いレベルでのモデルの抽象化は、生成されるソースコードを他のプログラム言語に容易に置き換えられることも意味します。

＜生成される出力ドキュメント(ソースコード)＞

５つ目になる最後の言語カテゴリーは、生成されるプログラム言語をベースにする方法です。これらの言語が簡単に構築できるのであれば、その言語の生産性と品質の向上は疑わしいです。クラスを四辺形で表し、そのクラスの詳細をテキストファイル内で修正するというモデル化手法から得られるものは殆ど有りません。

ただし、生成される出力が既にドメイン固有の言語になっている時にだけ、この方法も薦めることができるでしょう。例としては、特定のXMLフォーマットのプログラムを生成するようなケースがあります。XMLのスキーマは、言語のコンセプトと制約を特定する為の大量の情報を出力します。XMLメタファーに従えば、モデル化の段階において適格で妥当な設計が実現できます。また、グラフィカルなモデルを使用することで、たくさんのXMLの制限を克服することが出来ます。

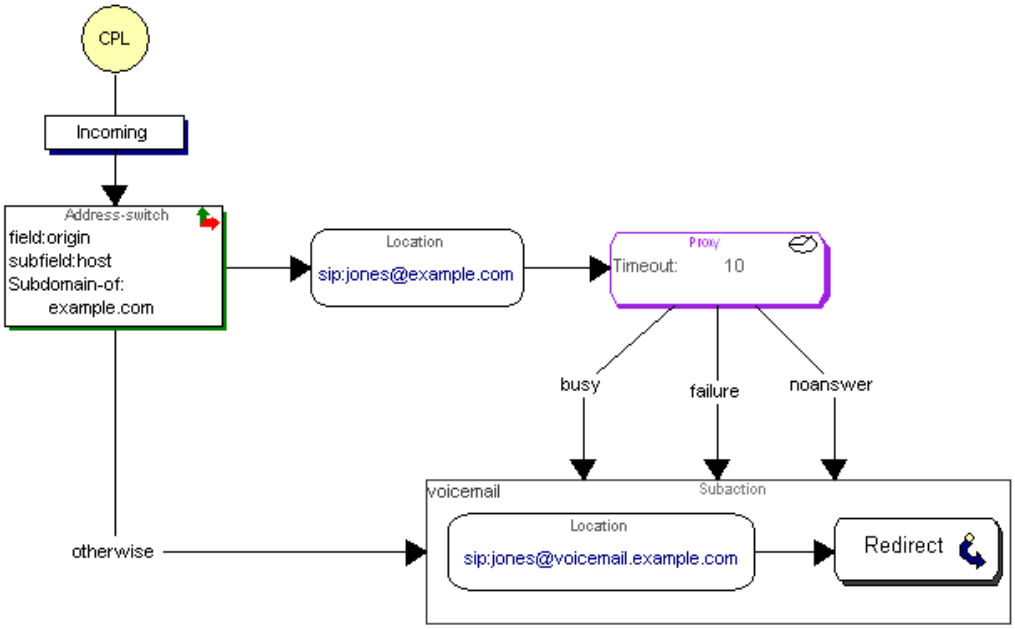

Figure5 IP プロセッシング：IP電話技術におけるコールプロセッシングの例、DSM言語は生成するべきXMLをベースにしています

この種類のDSM言語の例は、インターネット電話サービスを制御する、Call Processing Language(CPL)です。（Fugure5参照） 言語の構成は、IP電話サービスに不可欠な、信号動作、位置の選定、プロキシ機能を有しています。同じコンセプトが既にXMLのエレメントとして定義されており、モデル構造のプロパティー値がXMLエレメントの属性になっています。XMLの構成を生成するジェネレータを持つことで、有効な生産性と品質の向上を実現できます。Figure5に示されている言語を使用することで、（CPL/XMLを手書きすると簡単に発生してしまう様な）間違いや内部的な矛盾を排除することが出来ます。

言語仕様の完成

作成した言語をメタモデルの中に形式化することで、言語は完成します。メタモデルの形式は、使用するDSMツールに依存します。ただし少なくとも、言語のコンセプト、それらのプロパティー、言語のエレメント間の正しい接続、モデルの階層構造、正しいルールは定義できなくてはなりません。また、モデルの再利用や異なったモデルを統合する為の手法がサポートされていることは必須です。

メタモデルの作成は、独自の言語をモデル化することです。ドメインコンセプトを、多様な（オブジェクトのような）言語エレメント、それらのプロパティー、（オブジェクト間の結合状態を示す）リレーションシップや（オブジェクトが相互にどう振舞うかを表すロール）に結びつける(map)することで。幾つかの言語コンセプトは直接的に、その他は複数のドメインコンセプトの結合として定義できることが分かるでしょう。包括するコンセプトを選定するためには、使用するプログラム言語と生成する成果物が参考になります。従って、幾つかのコンセプトを定義した後直ぐに、その言語を試してみることが最良の方法です。多様なエディターが適宜使用可能で、それによって言語の定義のみに集中できるような理想的なツールがあります。これらのツールは、“言語が実際にはどう動くか？“や”如何に簡単に

モデルを作成でき再利用できるか？“の理解やテストが簡単に実行可能である為、アジャイルな作成を可能にします。これにより、作成した言語に間違いが発生するリスク或いは、作成した正しい言語が間違った動作をするリスクを最小化し、コードの自動生成の為の正しいマッピングを見つける上で、素晴らしい効果が得られます。

言語を実行可能な状態に保つ

DSM言語への移行は、特別な事ではなく普通の事になるでしょう。ドメインに対する理解は、（その専門家であるとは言え）言語を定義することでより深まるでしょう。なおかつ言語を利用していくことで、自身でのモデル作成や他のモデル作成者からのフィードバックを通して、理解が深まるでしょう。そして、ドメインへの理解がより深まり、言語の改良の可能性が見つかるでしょう。

言語の作成とテストが素早く行えることは、DSM言語を作成する上で生命線であると信じています。良い言語を作ることに集中することが必要なのであって、DSMツール内にその方法を実現することに気を取られるのが必要な訳ではありません。つまり、言語を定義した瞬間にテスト出来ることが最良なのです。これが、アジャイルに言語作成が行え、如何に迅速に言語を使えるかという点での大きな効力になります。大切なのは、言語の品質が向上されることであり、複数の開発者がその言語を使い始めた時にはその効果が何倍にもなることです。

モデル作成の候補者に早い時期に言語を渡すことで、彼らを巻き込むことと、早い時期のフードバックを得ることが可能になります。ドメインと同様に、言語も時間とともに進化し、修正の必要が出てくるでしょう。言語が進化をとめた瞬間が、その有効性が失われてゆく瞬間です。

選択の自由

新たな言語を全て定義することは、難しい作業と見られがちです。しかしながら、この作業は既に持っているドメインに対する知識を当てはめることだと一度理解すれば、作業が非常に簡単であると気づくでしょう。DSMを使用することは、アプリケーションやシステムの設計に見合わないような所定の図表や構文に落とし込むことを強要されない事を意味し、モデルを単なるドキュメント作成に過ぎない状態から飛躍させます。DSMは、どの設計言語がもっとも要求にあっているかを選択する自由を与えます。

開発者が常に取り組んでいる抽象化レベルを向上させる作業に真の機会を与えるだけでなく、言語を定義する開発者は、DSMを用いることで進化するドメインに対する専門的な知識を言語の中にカプセル化することが可能です。それによって、他の開発者は、その知識を覚えることなく、自動的に最良の手腕を追随出来ます。

良く出来たDSM言語を使うことで、チームは素早く最良の手腕を発揮できるようになります。DSMを使えば、必要な１００のコンセプトに対して７００の余分なコンセプトを提供するような、サードパーティーの万能な標準表記法を学ぶ必要がありません。それどころか、コンセプトを反映した１つのモデルと、既によく理解し、適応しているドメインのルールを利用することが出来るのです。多分最も重要なこと

は、モデルを、設計プロセスの一部に過ぎない単なるドキュメント生成から、真に実行可能なドキュメント生成に変えることでしょう。コードジェネレータの定義の詳細な内容は、別の文書に譲ります。

Dr. Juha-Pekka Tolvanen はMetaCase社のCEOです。モデル駆動型のアプローチとツール、とりわけ方法工学とメタモデル化手法に、１９９１年から深く関わっています。フィンランド、ユヴェスキュラ大学でコンピュータ科学の博士号を取得しました。現在、メソッド開発の為の国際的なコンサルタントとして活躍し、ソフトウェア開発手法における５０以上の論文を執筆しています。