Creating a Domain-Specific Modeling Language for an Existing Framework

Juha-Pekka Tolvanen, MetaCase 社　http://www.metacase.com

既存フレームワークを活用する専用のモデル開発環境（DSM）の構築

ドメインスペシフィックな言語は、コードライブラリやフレームワークの延長線上にあり、それらの共通コードを効率良く、容易に、堅実に使用することができる手法です。この資料では、ライブラリやフレームワークに対して専用のモデル開発環境を構築する方法について紹介します。

全てのソースコードをいちいち記述することは、余り意味がありません。いくつかの機能をコーディングすれば、コード内の似通った部分や、繰り返されるパターンに気付くことになるでしょう。多くの開発者にとって、同様の機能を繰り返し実装することに時間を費やすよりも、各機能・製品間のユニークな機能にのみ集中するほうが良いでしょう。分かりきったことを繰り返すのを避けることは、各エンジニアに対する助言にとどまらず、同僚が同じようなコードを書いている場合により顕著になります。

良く知られる手法は、共通部品を再利用可能なライブラリにすることや、フレームワーク内に置くことで各機能や製品の開発の基盤にすることです。その場合、アプリケーションエンジニアはフレームワークとその設計概念を学習し、クラスの階層を調べ、API を理解し、求められるプログラムモデルを学びます。フレームワークの使用法はドキュメント、アプリのサンプル、ガイドライン等でサポートされます。　問題は、殆どの開発者にとってフレームワークの使用は容易ではなく正しいコンポーネントを探すことが難しいと、研究結果、及び実践開発に於ける経験として報告されていることです。また、考慮されるべき相互依存性、同一機能を実現する複数の選択肢などの存在は、フレームワークを正しく使いこなす為の学習に時間を費やすことになるでしょう。

専用のモデル開発環境（DSM ）を用いれば、開発されるアプリケーションの抽象度を上げながらフレームワークの詳細を隠蔽することができるので、このような問題を解決します。専用のモデル開発環境（DSM ）では基本的な概念としてのフレームワークは利用可能なオブジェクトとなり、アプリケーションは高いレベルの概念でモデル化され、フレームワークを巧みに活かした実用的なコードが生成されるようになります。

この自動化は、特定のアプリケーション、フレームワークに対して、モデル環境、コードジェネレータを専用のものとすることで実現可能になります。自動化により、決まりきった作業を最小限に抑え、ありがちなエラーは未然に防ぐことになり、開発効率は根本的に速くなります。専用のモデル開発環境（DSM ）では、設計のルールやコンポーネントの使用法などもカバーできるため、フレームワークが正しく利用されることが保証されるようにもなります。全ての開発者は、熟練者が良く知るベストプラクティスを共有できるようになります。

DSM の例

どの様にして専用のモデル開発環境（DSM ）を構築するかを説明する前に、事例を紹介します。分かりやすいドメインとして、デジタル時計のストップウォッチ、時間設定と表示機能などのアプリケーション開発を例にします。これらのアプリは計算処理、アイコン表示、アラーム警鐘などの共通サービスを提供するフレームワーク上に実装されます。アプリを開発しフレームワークのサービスを呼出す為に、一定のプログラミングモデルが用いられるでしょう。

新しい機能を実装したり、既存のアプリを修正する前に、時計のドメイン内でどの様に動作するかのデザイン設計が必要です。これには、ボタン・アラーム・アイコン表示・状態・ユーザによるアクション、などの時計のドメインに対する条件やルールの適応が伴われます。DSM では、これらに対して極めて同様な概念をモデル開発言語に取り込みます。

以下、図１にそのような例を示します。時間に関する様々なアプリのデザインと開発用のモデル開発言語であり、この図は時間設定機能のデザインモデルです。ボタンを押すことで表示要素が点滅し、アクションによって時間が変更されます。

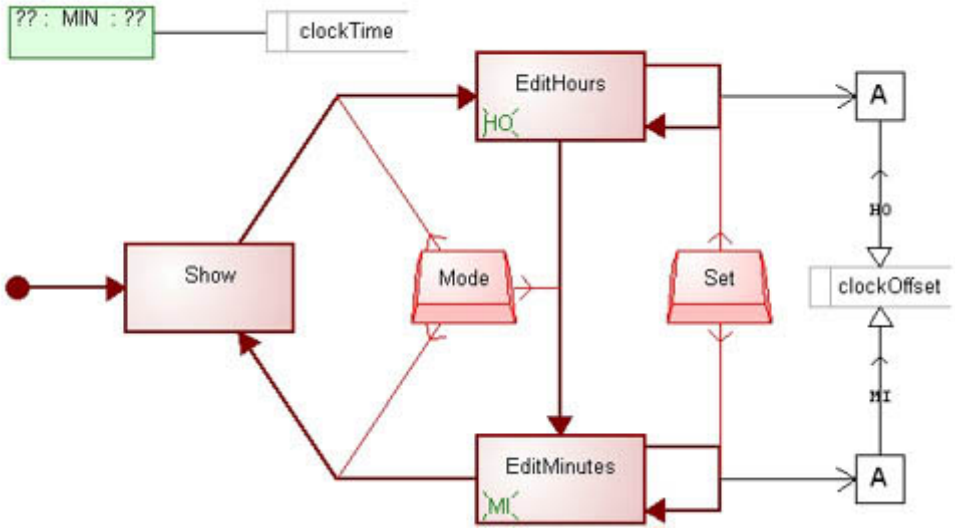

図１．時間設定機能のモデル

このモデル言語は、抽象度のレベルを上げてフレームワークのサービスの詳細を隠蔽していますが、時計の基本設計概念とそのフレームワークに明確に基づいています。この例では、開発者は単にアプリのデザインのみに集中することができます。このモデルから、フレームワークのサービスを用いて動作するコードは、自動的に生成されます。
単なるモデルではなく、このようなモデル開発環境があれば、多くの異なった時計のアプリを開発できるようになります。図２の世界時計のアプリも、同じフレームワーク、同じモデル開発環境を

用いています。追加のモードボタンにより、時間・分の変更切替えや、機能の終了などの、より高度な機能を搭載しています。緑色の表示機能では、clockTime　にオフセットを加えた時間を計算して表示できるようになっています。

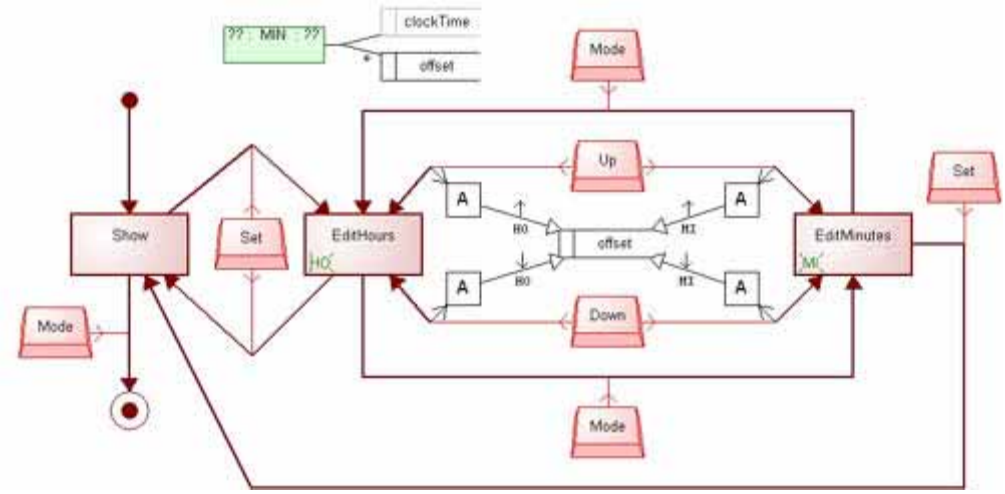

図２．世界時計のアプリケーションモデル

図３は、ストップウォッチです。図１．２ではなかった機能が使用されています。時間を変数としての計算機能、異なった状態に対するそれぞれの表示機能。ランニング中の表示機能は下側の緑色のオブジェクトで、現在のシステム時間からストップウォッチが開始された時間を差し引いて表示します。

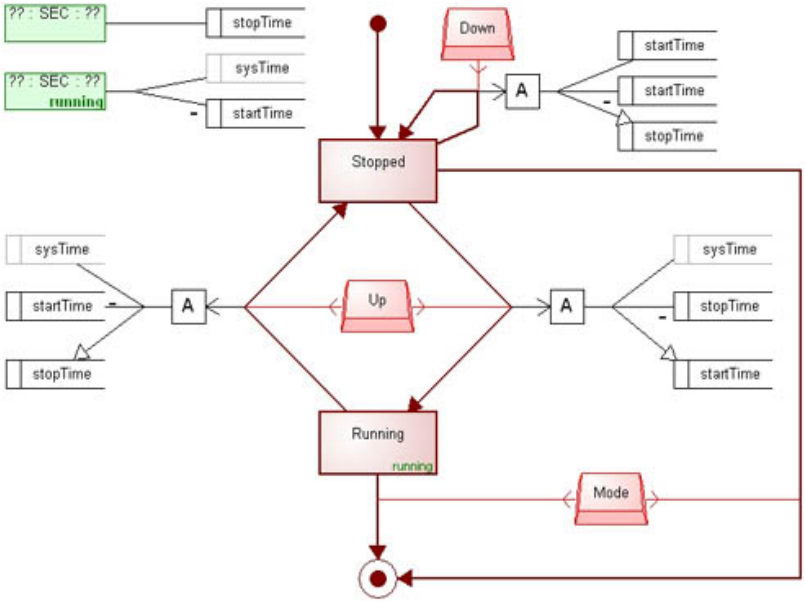

図３．ストップウォッチのアプリケーションモデル

これらアプリは、ソースコード開発と同様に追加・修正ができるようになっています。改善が必要であれば、モデルを編集します。図４の例は、ストップウォッチにラップタイム機能を加えたものです。（Laptime というパラメータを状態メタモデルに定義し、処理を他のモデル部品で追加）

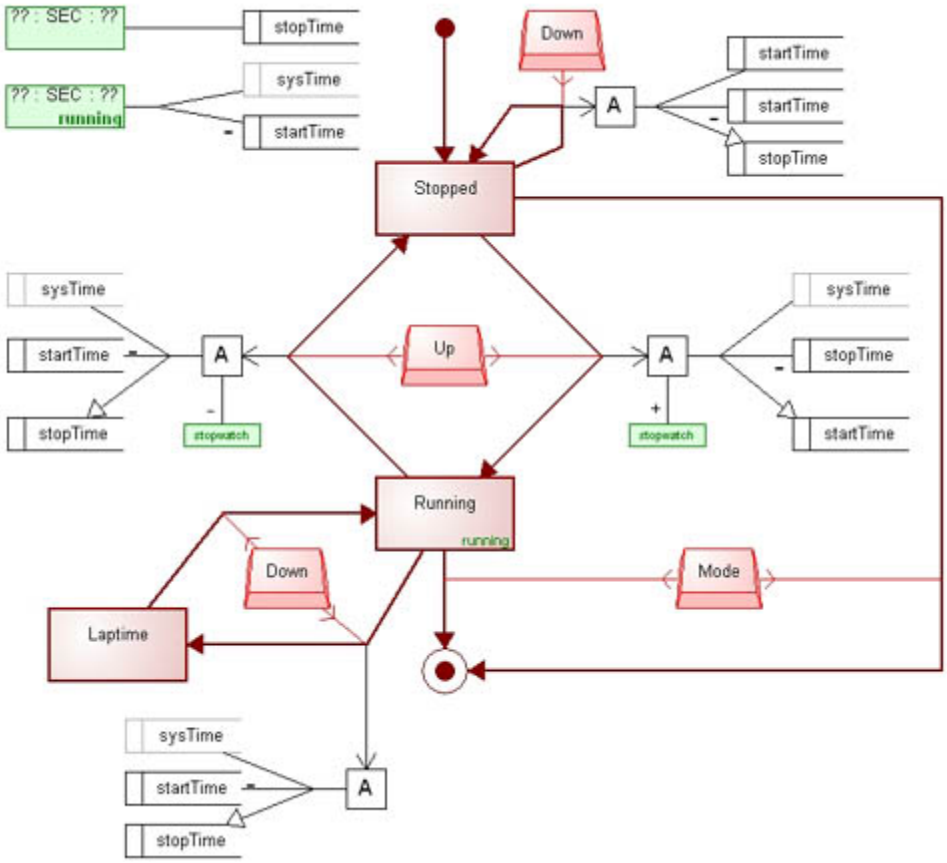

図４．ストップウォッチにラップタイム機能を追加

では次に、どの様にして専用のモデル開発言語を定義するかを解説しましょう。

専用のモデル開発環境（DSM）構築のステップ

たいていの場合、少しずつ追加するようにアプローチすることが最良策です。最初に小さな部分を設定し、それを用いてモデル化し、ジェネレータを定義して、モデル言語に修正を加え、さらにモデル化して、といった手順です。それらがフレームワークで動作できるようになってから、他の部分をカバーするようにモデル言語を拡張し、フレームワークも完成させていきます。これはフレームワーク開発において、そのコンポーネントが完全に用意されていないような場合と同じ取り組み方です。

Creating a Domain-Specific Modeling Language for an Existing Framework

Juha-Pekka Tolvanen, MetaCase 社　http://www.metacase.com

モデル言語とコードジェネレータの設定は、以下の４段階に分解することができます：

1．アプリの抽象概念、およびそれらがどの様に連携するかを特定する
2．モデル言語のコンセプト、ルールを定める（メタモデル）
3．言語に対して視覚的な表記部品を設ける
4．モデルのチェック、コード生成、ドキュメント生成の為にジェネレータを定義する

通常プロセスはこの手順で進行しますが、これらの反復・繰り返しも可能です。ここでは、手順２．モデル言語の設定についてフォーカスします。手順１．抽象化作業は、おおむねフレームワークのコンポーネント、ライブラリとして行われるでしょう。つまりフレームワークによって、うまくモデル言語を構築することができるようになります。残された作業は、フレームワークとそのサービスをアプリケーション開発者が簡単に使用出来るような仕組みを構築することです。（これが DSM です！）

モデル言語を定義する場合の注意は、コードやフレームワークの API コールなどを視覚化するようなモデルにしないことです。その代わりに、プログラム言語よりずっと高いレベルの抽象化を探し出すことです。さもなければ、プログラムのドキュメントとしてのフローチャートを作り直すだけの結果に陥ることでしょう。フレームワークに加えて、その上で動作するサンプルのアプリがあればどのようにフレームワークが使用されるのかを評価しやすくなるでしょう。サンプルのアプリは、コードジェネレータに対する良い情報にもなり得ます。

モデル言語の抽象度を上げることで生産性が向上できるだけでなく、フレームワークを正しく用いる作法を施行できるようにもなります。これこそ、全ての設計者が望んでいることです。これらのルールやガイドラインが正しくモデル言語に定義されることで、フレームワークから外れたアプリや機能などは起こり得ないようになります。また、各アプリ開発担当者は、組織内の旧態依然としたデザインガイドラインを継続的に参照することから開放されるようになります。

モデル言語の定義

モデル言語の定義は、モデリングのコンセプトと、それらの間で取りうる接続を特定することです。一般的な出発点は、アプリケーションについて打ち合わせる場合に用いてきた何らかのモデル（スケッチのような）を利用してみることです。ただしそれは、実装向けのモデルではありません。例えば、UML のクラス図のようなモデル言語にすることは避けるべきです。これらは実装に対して視点を置いており、製品ごとの固有の課題には殆ど配慮がありません。

単純なケースであれば、フレームワーク内に全ての取りうる機能が用意されていることで、モデルエレメント（部品）のバリエーションはパラメータで指定することができるでしょう。(例えばデジタル時計のボタンやアイコンに関しては、このアプローチを用いて複数からなるものをまとめています)　ただ一般的にはこれほど単純ではなく、パラメータ間に相互依存性がある場合には、取りうるバリエーションを指定するために最低でもウィザードやデシジョンテーブル（条件一覧表）などが必要になるでしょう。

ただ殆どのバリエーションはパラメータの一覧から選択できるほど簡単ではありませんので、どちらの方法論も全てにおいて適切な選択肢とは言えません。例えば時計がアイコンを持つかどうかを検討する際は、アイコンが使用された状態をも知る必要があります。また、パラメータの一覧を用意する方法は、全てのフレームワークのコードが全て実装済みである場合には適応可能ですが、まだアプリケーションが実装されていないような典型的な状況下では役には立たないでしょう。

さらに、バリエーションが静的な構成から動作を記述するコード、もしくは振舞いを実現するコードにまで及ぶ場合も、パラメータリストや機能の品揃えでは十分ではありません（例えば、５階層のブーリアンを用いてもせいぜい３２種の組合せしかできないでしょう)。これらの場合のモデルはパラメータのリストでまとめられるほど簡単ではなく、それらに関わる全ての要素は予め理解されている必要が有るでしょう。
以上のことから、モデリングする場合には新しい要素やリンクを新しく構成できるようにされているべきです。そのようなモデル言語は、デジタル時計例で紹介しました。(図３：状態に依存した時間計算と表示)（図４：ラップタイム機能の追加。状態によって Down ボタンから起動される処理が異なっている）このようなモデル言語には、バリエーションや新しいアプリケーションに対する制限がありません。

周知の処理からモデル化する

時計のアプリケーションではイベントドリブンな状態遷移をベースとしている為、ステートマシンをモデル言語のベースにすることにしました。このモデルにおける各種処理は、ドメインコンセプトとルールで肉付けされます。

図５は、ステートマシンの基本的な定義を図解しています（メタモデルで)。このメタモデルでは、各ステートマシンは１つのスタートオブジェクトを持ちステートへの遷移が先導されます。ステートオブジェクトは２つの属性を持っています：(モデル内でのユニークな名前とテキストによる説明)　ステートは他のステートへの遷移、もしくはストップオブジェクトに接続されます。(これらの遷移は同じ役割のロールであるため、バインディング用のオブジェクト内へ一緒にまとめられます。)

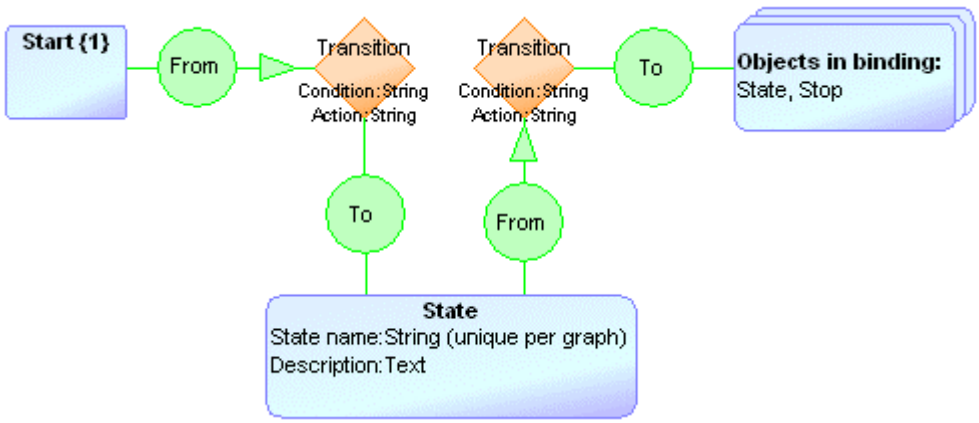

図５．ステートマシンの基本メタモデル図

さらにモデル言語はルールを持ち、どのようにモデル化され修正できるかの制約を持たせることができます。これらのルールは言語のコンセプトと合わせてメタモデル内に定められます。例えば図５．のようなステートマシンでは、スタートステートに入ってくるイベントは無く、各状態遷移は１つのスタートステートしか持ち得ない、など。

ドメイン固有のコンセプト、ルールで言語を拡張

時計固有のコンセプト、ルールを用いてモデリング環境を肉付けします。これはドメインのコンセプトやフレームワークのサービスをモデリングの各要素にマップ（関連付け）することです。それらの一部はモデル言語内のオブジェクトとして、他はオブジェクトの属性、リレーションシップ、サブモデル、他のモデルへのリンクなどで得られるでしょう。

例えば時計の例では各アプリのステートは時間を計算し表示する機能を持っています（現在時間の表示、オフセットを加えた世界時間表示など)。他の時計の特性として、ボタンが押されることで状態の遷移が発生するなど。

ボタンの系統だったコンセプトはモデル言語の仕様として追加されました。図６では、ステートに２つの属性を追加して拡張されたメタモデルです。時間の修正を施すステートでは、時・分・秒などの修正される部分が点滅表示されます。

時計のアプリをより詳細に記述するためにトランジションが修正されました。ユーザにより時計のボタンが押されることでトランジションが始動され、遷移中にアクションが実行されます。これらは必須では有りません（図６のメタモデルにはアクション、ボタンのコンセプトに関わるトランジションのロール０、１があります）

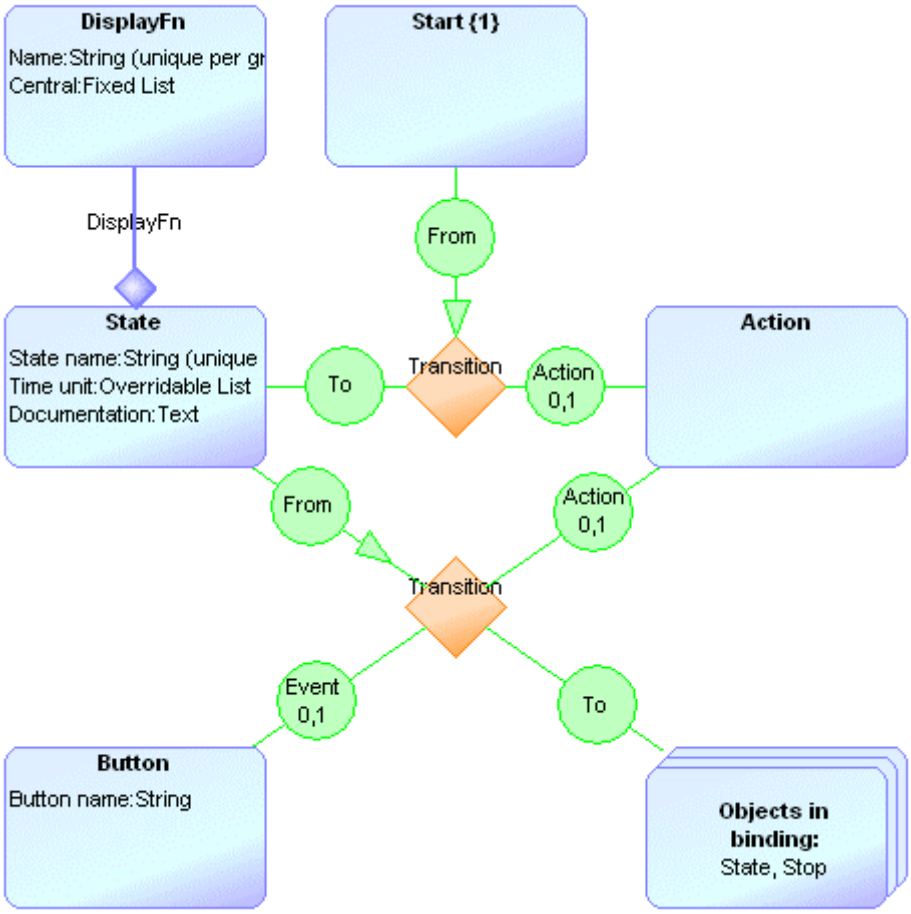

図６、ドメイン固有のトランジションで拡張されたステートマシン

同様にその他のドメイン固有のコンセプト、フレームワークのサービスがモデル言語の仕様として追加されました。例えばアイコンの使用のためのサービスはフレームワークから提供されます。これは、以下図７のメタモデル内で、アクションとアイコンの間のリレーションシップで定義されています。

リレーションシップは属性を持ち、どのアイコンサービスが使用されるか（on, off）を指定します。他のフレームワークのサービスとしては、アラームの警鐘などがあります。図７のメタモデルでは、変数を介して時間の設定や計算を行うフレームワークのサービスが含まれ、どのように時間が計算されて表示されるか、アラームがセットされるかを記述しています。

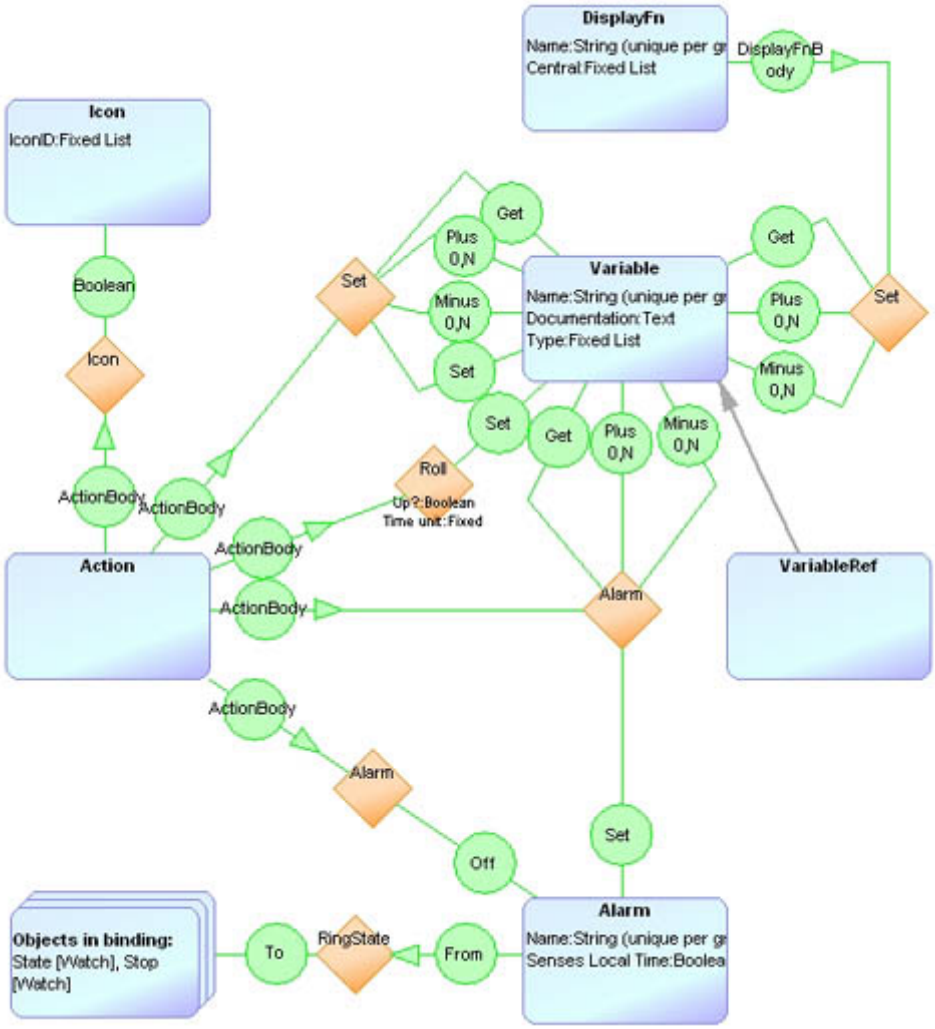

図７、時計固有のアイコンを加えてステートマシンを拡張

このメタモデルの仕様はフォーマルであり、モデル言語のベースになります。実際、図１～４のモデルは、このメタモデルをベースにしています。それゆえ、このメタモデル内で設定されたコンセプトやルールは、図１～４のデザインモデルのベースになっています。

視覚的な表記の設定

しかしながら純粋なメタモデルのみでは十分ではなく、ダイアグラムなどの視覚的な表記も必要です（マトリクスやテーブル、テキストなどの表現も場合によって）。視覚的な表記に最もふさわしい素材は、アプリケーションごとのドメインです。親しみのあるアプリケーションごとのシンボルを用いれば、読みやすく、覚えやすく、理解やチェックもやりやすくなります。ホワイトボードやプレゼンスライドに描かれる自由なスケッチを参考にすると良いでしょう。これに対してアプリケーションのコンセプトを当てはめて、視覚的に表現します。DSM には、これらのコンセプトや表記を

モデル言語として取り入れるのです。

時計のサンプルでは、ベーシックなステートマシンの表記に、ボタンやアラーム、アイコンなどのドメインコンセプトで拡張を加えました（図１～４）。また色分けによって異なるエリアを区別するようにしました。緑色で表示系（アイコン、画面表示）、赤色でコントロールパス（ボタン、トリガ、アラーム）、黒色で根底をなすデータモデル（時間操作など）。エンジ色は、データモデルとコントロール両方の側面に用いました。

実行して評価

メタモデル、もしくはその一部が定義されればサンプルのアプリケーションでテストすることをお勧めします。その為には、定義された言語のモデリングを行う為のエディタを含めたツールが必要になるでしょう。同じツールで、言語の定義と利用が行えるのが最良です。これにより、アジャイルに言語の定義が行えます（言語を利用しながら言語を修正できます）。

しかしながら、ツールの機能は、メタモデルをベースにしたエディタのみでは足りません。ツールには、メタモデルを定義する為のガイドや、不完全な言語定義（例えば、不正な組み合わせがある等）を認識する機能が必要です。また、開発者間の言語バージョンの共有や言語の変更をベースにした既存モデルのアップデートにも対応しなければなりません。殆どの企業では、新しい言語バージョンが出来るまで、数週間、或いは、数ヶ月待ち、それからモデルを書き換えるなどの余裕は無いので、この機能は実用上非常に重要になります。

これら機能を提供するツールは既に存在しており、熟練者が各ドメイン専用のデザインモデリング言語とジェネレータを設計し、アプリ開発者がこの DSM ツールを用いて製品モデルから実製品用のコードを生成できるようになります。
この手のツールを自作する方法も有りますが、簡単な試作のエディタを作るのにも１人年以上必要であり、それを開発者が受け入れるような、信頼性のあるツールにするのに更に多くの作業が必要であるでしょう。

結び

フレームワークを上手に拡張し使いこなすことが、専用のモデル開発環境（DSM）の意図する所です。専用のアプリケーション領域にフォーカスし、似通ったアプリを上手く生成させる狙いはフレームワークも DSM も同じです。必要なのは、ドメイン専用の言語を構築し、ジェネレータを定義することです。これらの作業に対する投資はすぐに回収できます。例えば、時計のサンプルでは 2 日間でモデル言語が出来上がりました。別途、5 日かけて Java を勉強してフレームワークを用意し、

1 日でコードジェネレータを用意しました。これらの作業で最初の時計機能は用意でき、その後は新しい時計の開発は 20 分でテストまでできるようになりました。
相当な開発スピードの向上が得られるだけではなく、DSM を用いてフレームワーク開発を上手に行えば、

- フレームワークに対する定型的な作業が最小限に
- フレームワークの複雑さを隠蔽
- DSM に組み込まれたルールがより良く踏襲される
- 生成されるコードは高品質（熟練者の作法が共有される）
- フレームワークに対する変更が容易で、それに伴うアプリの再生成もシンプルに実現できる

これらの恩恵は、従来型（フレームワークに関する仕様書を読み、ライブラリのコンポーネントを調べ上げ、皆が共通の基本設計概念に則ることを期待する）の手動によるプログラミングでは簡単には得られないでしょう。もし、フレームワークを用いている、もしくは準備しているのであれば、DSM による専用のモデル環境を検討するべきでしょう。興味深く、大いなる楽しみをももたらすことにもなるでしょう。

ツールの評価版には、時計のサンプルも含まれておりすぐにでも評価していただけるようになっています。http://www.metacase.com/ja/download/index.html

FUJI SETSUBI
富士設備工業株式会社 電子機器事業部
〒591-8025 大阪府堺市北区長曽根町1928-1
Tel: 072-252-2128　www.fuji-setsu.co.jp