ドメイン固有のモデリング言語がどのように変動性に対処するか：23 の事例調査

How Domain-Specific Modeling Languages Address Variability: Investigation of 23 Cases

Juha-Pekka Tolvanen, Steven Kelly //MetaCase //Jyväskylä, Finland //(jpt,stevek)@metacase.com

https://www.metacase.com/papers/How_Domain-Specific_Modeling_Languages_Address_Variability-cameraReady.pdf

概要

ドメイン固有のモデリングは、ドメインの概念でソリューションを直接指定することにより、抽象化のレベルをプログラミングのそれから引き上げる。 プロダクトライン内では、変動性を特定して、共通性とともに最終製品を生成するために、ドメイン固有のアプローチが適用される。 このような派生製品の自動生成は、特定のプロダクトライン（多くの場合単一の企業内）向けのモデリング言語とジェネレータを作ることで可能になる。 この資料では、ドメイン固有のモデリングの産業界の事例から、どのような種類の再利用およびプロダクトラインアプローチが適用されているかを検証する。 これは 23 の業界事例と、そこで作成された言語とモデルの実証分析に基づいている。 この分析により、言語のサイズと、再利用およびプロダクトラインアプローチの適用方法における多種多様な共通点を明らかにした。

CCS CONCEPTS

·Software and its engineering→Software product lines; Model-driven software engineering; Domain specific languages; Abstraction, modeling and modularity; System modeling languages; feature modeling.

KEYWORDS

Domain-specific language, domain-specific modeling, product line variability, product derivation, code generation

ACM Reference Format:

Juha-Pekka Tolvanen and Steven Kelly. 2019. How Domain-Specific Modeling Languages Address Variability in Product Line Development: Investigation of 23 Cases. In23rd International Systems and Software Product Line Conference - Volume A (SPLC '19), September 9–13, 2019, Paris, France. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336294.33363161

1 はじめに

ドメイン固有の言語とモデルは、ドメインの概念でソリューションを直接指定することにより、抽象化のレベルをプログラミングのそれから引き上げる。これはプロダクトラインの開発に特に適している。 変動性と関連するルールが言語の一部として、バリアントの仕様作成に使用されるためである。そして、この高レベルの仕様から最終製品が生成される。 この自動化は、言語とジェネレータの両方を、特定のプロダクトラインのみの要件に適合させることで可能になる。 プロダクトラインに対する言語ベースのアプローチは、より大きな変動スペースを管理し、自由度と柔軟性を高め、パラメータテーブルやフィーチャモデル（Czarnecki2000）などの他のアプローチでは不可能なバリエーションを作成できる。

Domain Specific Specific Modeling（DSM）は広く採用され（Sprinkle2009、Tolvanen2005）、Product Line Hall of Fame にも、それらを使用する産業界の事例があります。しかしながら、どの様な言語が作成

されて適用されたかについての調査はほとんどありません。 プロダクトライン開発に使用される言語を分析する研究は、フィーチャモデリング言語、その拡張機能（Zaid2010、Cognini2015、Sousa2016）およびそれらの比較（Acher2012、Czarnecki2012、Wahyudianto2014）に焦点を当てる傾向があります。 ドメイン固有の言語が調査される場合は、特定の状況およびプロダクトラインに対しての、異なる言語構造の評価と比較がフォーカスされました（例（Preschern2014、Mewes2009））。 多数のドメイン固有言語を分析する研究では、言語構造の調査（Tolvanen2005）、または良くない事例（Kelly2009）に焦点が当てられています。

この調査では、企業がプロダクトライン開発用に作成した言語の種類を調査します。 特に、これらの DSM 言語がバリアントを定義し、これらのバリアント定義を再利用する方法に焦点を当てます。 私たちは経験的なアプローチを取り、23 の業界事例とそこで作成された DSM 言語を分析します。 分析では、言語定義（メタモデル）、目的とする出力（コードの自動生成など）、言語作成に関与する人々、およびモデルの外部と内部の両方で言語によって有効化された再利用を調査します。 この調査から、言語のサイズは大きく異なり（最大のものは最小のものより 14 倍大きい）、バリエーションを管理するためにさまざまな言語構造が、事例ごとで使用されることがわかりました。

次のセクションでは、各プロダクトラインと言語の分析方法について説明します。そして Section 3 では、プロダクトライン開発をサポートするためのさまざまなアプローチ（モデリング言語構造とプロセス）の例を説明します。 Section 4 でデータを分析し、このアプローチの分類を評価し、Section 5 で結論をまとめます。

2 調査されたプロダクトラインについて

この調査は、プロダクトライン開発に適用されたドメイン固有のモデリングの 23 の業界事例の分析がベースです。 過去 15 年にわたって、著者が言語の定義に関わることができた事例を選択しました。多くの場合、言語とジェネレータの作成をサポートするコンサルタントとしての役割で、全ての作成を任されたのではありません。 そして、すべてのモデリング言語とジェネレータは MetaEdit + （Kelly1996、MetaCase2018）で実装されました。 ただし、認識された言語パターンは特定のツールに限定されません。 重複を避けるため、2005 年の論文（Tolvanen2005）と異なるケースのみを選択しました。

Table 1. Product line DSM cases analyzed

#	Domain	Targets	By	Size	Use	Approach
1	Consumer electronics	C, HTML, Docs	1		3	1
2	Industrial automation	PLC, GUI, DB schema, net config, deploy	1	165	5	1
3	Enterprise applications	C#, DB schema	1		6	1
4	Railway signaling	Simulation, XML	1	291	6	1
5	Signal Processing Systems	Matlab, simulation, XML	1		4	1
6	Oil drilling	Cost calculation, documentation	1		3	1
7	Big data applications	Java, JSON, CQL, SPARQL, SQL	2	397	4	1
8	Printing process	Ruby, XML, Docs	3	55	4	1
9	System performance	Gherkin, HTML, Docs	1	145	3	1
10	Consumer electronics	JSON	3	72	2	2

11	Telecom service	XML	1	61	2	2
12	Medical	XML, audit documents, change history	1	63	1	2
13	High-level synthesis	System C	1	450	3	2
14	Radio network	TTNC-3, simulation/animation	1		5	2
15	An automotive system	System specification	3	62	3	2
16	Database applications	Java	3	46	1	2
17	Consumer electronics	C, localization, docs	1	403	6	3
18	Automotive architecture	Simulink, ISO26262 documents, AUTOSAR	3	652	6	3
19	Telecom	C, build automation	2	109	3	3
20	Insurance	Cobol, DB schema	3	234	6	4
21	Aerospace	C#, XSD, JSON, API	2	121	1	5
22	Automotive ECU	Python, JSON, Test document, change history	3	64	5	5
23	Software testing	Propriety format of state machines	3	317	6	6

すべての言語定義は、メタモデリングによって自由に作成されました。既存の言語に対するカスタマイズや拡張、またはプロファイルなどの制約に限定されないということです。言語の定義に完全な自由が得られ、また言語に関わるツールも適用または作成される言語構造を制限しません。モデリングとプロダクトラインに関する文献には異なるアプローチも見られます。（Acher2012、Czarnecki2012、Wahyudianto2014）で提案されているように、フィーチャモデリングのサポートを直接適用するか、拡張することができます。同様に UML をステレオタイプとプロファイルで拡張することや、（Mewes2009）のようにメタモデルベースとプロファイルベースの両方を定義することもできます。

事例は、様ざまな業界の多様なプロダクトラインをカバーするように選択しました：家電、データベースアプリケーション、自動車および産業オートメーションシステムまで。 表 1 は、これら事例を、各ドメインおよびその他の特性別に、プロダクトラインの側面に焦点を当てたモデルレベルの大まかな順序で要約しています。

すべての事例で DSM は開発の自動化に活用されました。計画、設計、またはコミュニケーションのサポートだけではなく、モデルチェックを実行し、モデルから期待される出力を生成することでプロダクトライン開発を自動化しました：主に生成されるのはコードですが、Simulink や AUTOSAR のモデル、形式手法に用いられるフォーマルなモデル、コンフィグレーションファイル、監査用の仕様およびドキュメントなどもあります。 生成される主な出力は、表 1 列の **Targets** にリストされています。

表 1 の“**By**”列は、言語が誰によって開発されたかを示します。ドメインの知識や言語開発の経験などの違いです。

1.社内組織で開発
2.言語作成の専門知識を持つ外部コンサルタントによる開発
3.両方

DSM 言語のサイズは大きく異なり、最小のものは 46 の言語要素で、最大のものは 652 の構成要素で構成されます。 これらの値（入手できたもの）は、表 1 の“**Size**”列に示されています。 比較情報として、同じ GOPPRR メタ-メタモデル（Kelly1996、MetaCase2018）で実装された UML 2.5（OMG2017）には 247 の要素がありました。（GOPPRR は MetaEdit +のメタモデリングフレームワークです）

言語がどの程度活用されているかも事例によって異なります（表 1“**Use**”列）。

1.言語はまだ定義中
2.安定した言語でサンプルモデルを作成
3.テストとしての実際のケースの重要なモデリング
4.実パイロットプロジェクト
5.本番使用
6.長期の生産で使用

プロダクトライン内で開発されたバリアントの数など、言語の使用状況に関するデータは、すべての事例で得ることはできませんでしたが、わずかなものから数百のバリアントまで、様々でした。言語の作成と保守に必要な投資や労力については、（Tolvanen2018）で公開されており、利用するツール間の比較もあります （Kouhen2012）。DSM を実開発に使用することの成功と影響の詳細については、（Tolvanen2016、Whittle2014）を参照ください。

表 1 の最後の列“**Approach**”について、以下のセクションで説明します。

3 言語はどのように変動性に対処したか

バリエーションを指定するためのアプローチの分類は、主に筆者が利用可能な言語定義から集めました。それゆえ言語の定義は MetaEdit+で直接調査することができ、サンプルバリアントモデルが作成されました。調査のために我々はまた、作成されたバリアントモデルを分析し、言語やジェネレータの作成に関与したコンサルタントや社内の開発者から、我々の理解を確認しました。

すべての言語に共通するのは、変動性を表現して、ほとんどの共通性を既存のレガシー、プラットフォーム、コンポーネントなどに残したことです。多くの場合、ドメインフレームワークが言語とともに作成され、さらに共通性が抽出されます。これらの共通部分は、ジェネレータを介して統合されます。

事例とその言語の分析により、変動性と再利用に対処するための 6 つのアプローチを特定することができました。分類は主に言語定義に基づいていますが、プロセスの問題も考察され、一部のアプローチでは特定のツールサポート機能が必要になる場合があります。識別の主な基準は、作成されたモデルがバリアント間で再利用されるかどうか、およびその再利用がどのように行われて維持されるかでした。 バリアントの特定の共通に再利用される部分が集中的に維持され、新しいバリアントの作成時にすべてのモデラーに提供される場合、これらは「コア」モデルと見なされました。

事例からのデータにより、以下の分類を得ました（表 1 の最後の“Approach”列）。

1.各モデルとその要素は単一のバリアント用
2.モデルやモデル要素を複数のバリアントで再利用
3.各バリアントでのモデルやモデル要素の再利用をマーク/フィルター/変更
4.コアモデルとバリアントモデル
5. コアの制限されたバリエーション用のコアモデルと言語
6.マルチレベル：モデル要素は言語要素になる

このアプローチのリストは完全ではありませんが、適用される典型的なアプローチを明らかにします。いくつかは他よりも一般的です。 順序付けは、アプローチがプロダクトラインの明示的なサポートに重点を置いているという私たちの解釈に従います。 私たちの経験では、後のアプローチに初期のアプローチも含まれていることが多く、言語が開発および使用されるにつれて、アプローチの規模に沿ってさらに移動することがあります。 次のサブセクションでは、各アプローチをより詳細に説明し、実際の例を使用して説明します（許可されている場合は実例を使用）。

3.1 各モデルとその要素は単一のバリアント用

ドメイン固有言語を適用するための古典的なアプローチは、（Weiss1999）によって、次のように説明されています。：言語は変動部分の記述に焦点を当て、共通部分はフレームワークやコンポーネントなど言語の範囲外で定義される。バリアントの派生フェーズで、ジェネレータがモデルを読み取り、バリアントデザインをレガシーコード、フレームワークなどの共通部分と統合する。

このアプローチに従う言語の主要な特徴は、モデラーが一度に 1 つのバリアントの開発に集中することです。モデルに加えられたすべての変更は、単一のバリアントのコンテキスト内で行われます。 このアプローチは組織的に明確です。製品開発チームは変動性のすべての側面を所有し、テスト、バージョン管理などに単独で責任を負います。このような言語は、オペレーターごとのテレコムサービス、工場ごとの産業オートメーションシステム、顧客ごとのサービスなどの場合に一般的です。

Fig. 1 と Fig. 2 は、これを実際の例で示します（Preschern2012）。 養魚場向けの自動化システムを製造する会社は、各顧客向けのシステムに対して個別の仕様を作成します。 システムの一部として提供されるすべての Aerator（エアレータ、通気装置）については、バリアントに特定の値の設定（または初期値を使用）が、言語の機能により行えます。例えば、 電圧値や、どのような種類の酸素が使用されるかなどの設定です。言語によるバリエーション空間の設定は、Fig. 1 のように、プロダクトラインのパラメーター設定とみなされます。

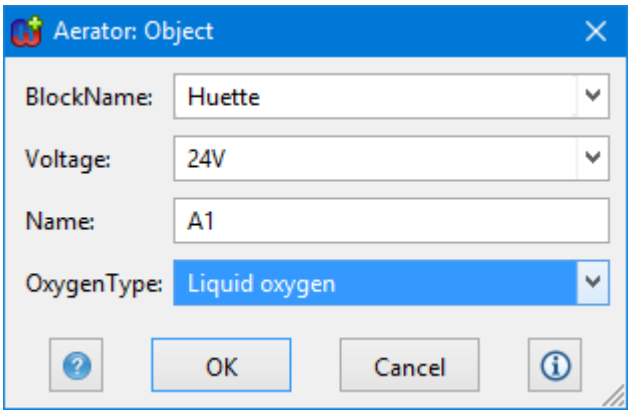

Figure 1. Setting values for a variant of Aerator

ただ実際には、すべてのバリエーションはそれほど単純ではありません。変動の単位は、言語で表現できるプロダクトラインの任意の側面です。たとえば、Aerator に関連する追加のバリエーションは、その場所、養魚池との関係、ネットワーク、電源などです。DSM 言語は、このようなリッチなバリエーションも利用できます。Fig. 2 は特定の Aerator が変動部分として表記される自動化システムの一部です。たとえば、青い楕円は魚の池であり、その変動性の 1 つは場所です。 このバリアントは、顧客固有の他の機能（照明、給餌、Ph レベルの監視、濁り、温度など）、および関連する構成を持つネットワークも指定します。 場所、順序、接続、および振舞いは、ウィザードやパラメータテーブル、またはフィーチャモデルで表現するのが困難または不可能なバリエーションですが、DSM ならうまく表現することができます。

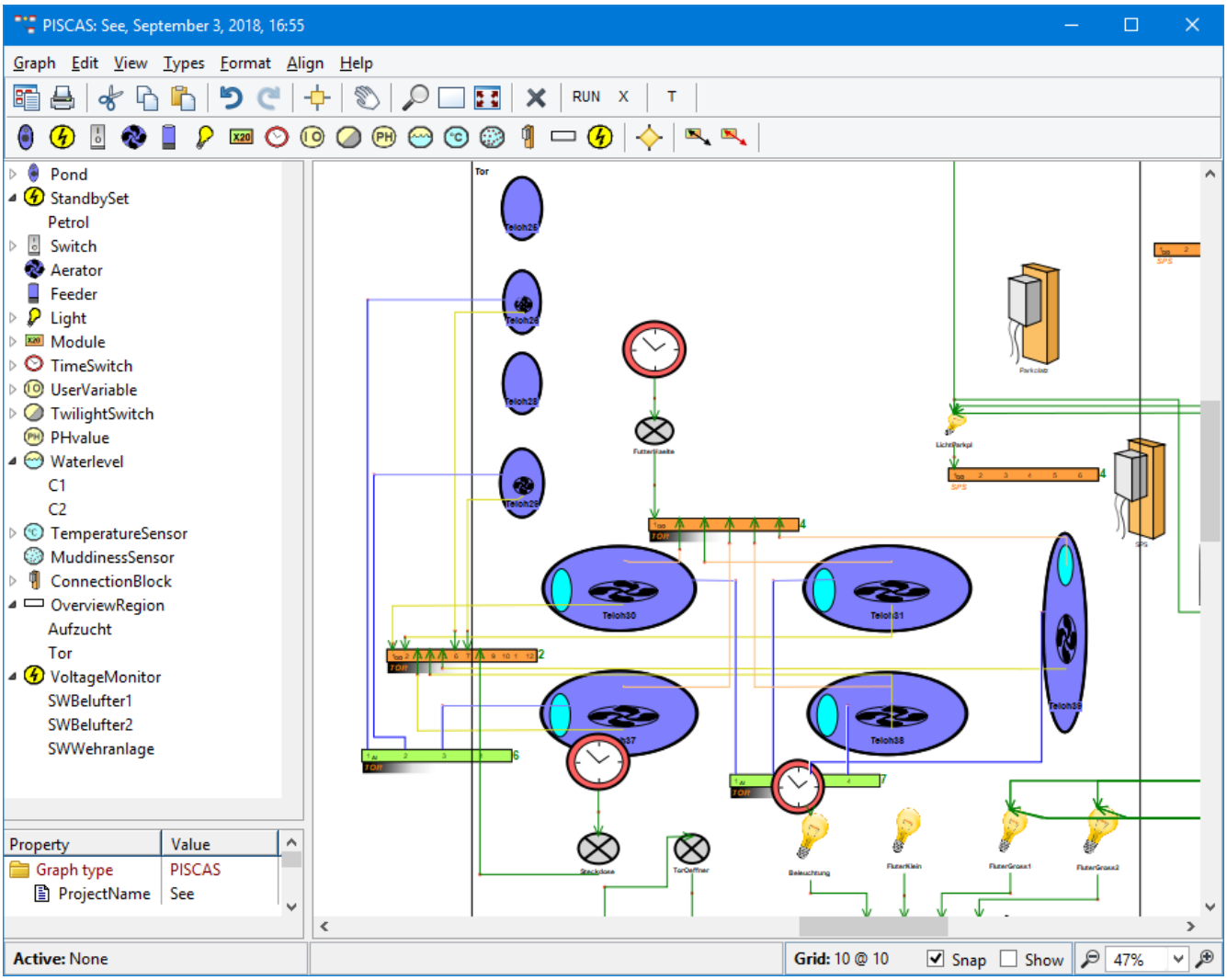

Figure 2. Variant of fish farm automation for a given customer

Fig. 2 のようなバリアントモデルから、同社は自動化システムの PLC コード、永続データを保存するためのデータベース定義、特定のセットアップのネットワーク構成、設置ガイドライン、材料のニーズなどを生成します。池の場所は、自動化システムを監視および制御するためのユーザーインターフェイスを生成するために使用されます。

単一のバリアントに焦点を当てた言語は、バリアント開発チームが独立して作業し、あるバリアント用に作成された機能を他のバリアントと共有する必要がない場合に役立ちます。もしそのようなニーズが生じた場合（かつ適切な言語サポートがないなら）、多くの場合、残された唯一の選択肢はクローン＆オウンで、元のバリアントモデルのコピーに対して行われた変更の追跡・管理です。

3.2　モデルやモデル要素を複数のバリアントで再利用

さらにバリアントが開発されると、多くの場合、既存の作業を再利用する必要が生じます。あるバリアントにすでに定義された機能が、他のバリアントにも有用である場合など。 クローン＆オウンを回

避するために、サブモデルまたは要素を複数のバリアントから参照させます。（モデルを階層化して、サブモデルの再利用を許可することは Approach 1 にもありますが、再利用は常に単一のバリアント内で行われます）Approach 1 と同様に、多くの場合、最上位モデルの名前は事実上バリアント名です。

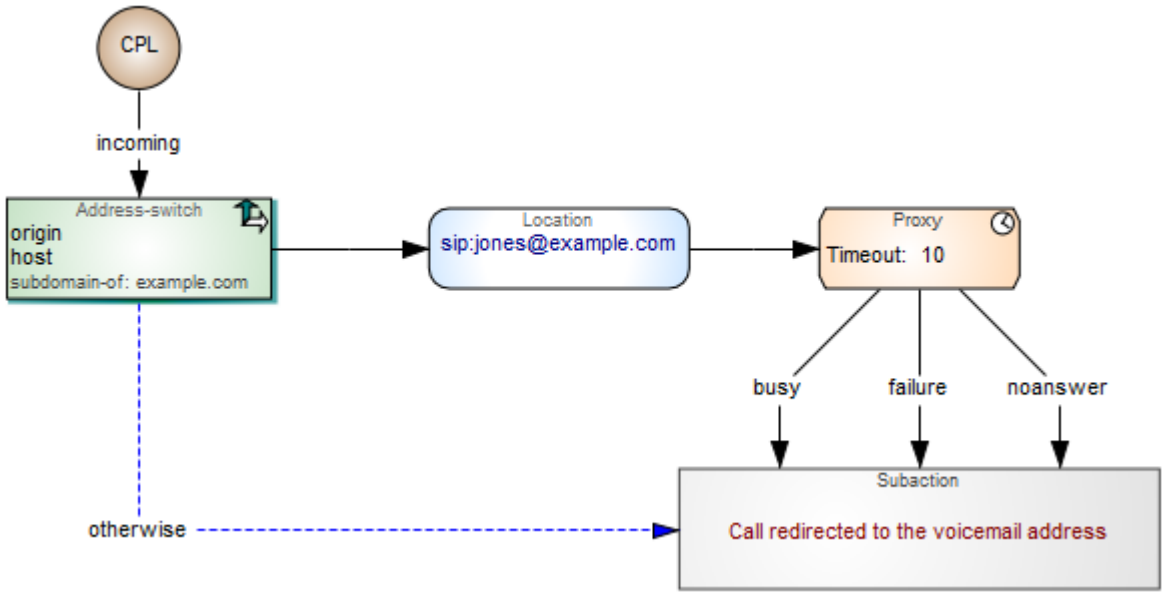

Figure 3. Location-based call redirection to voice mail

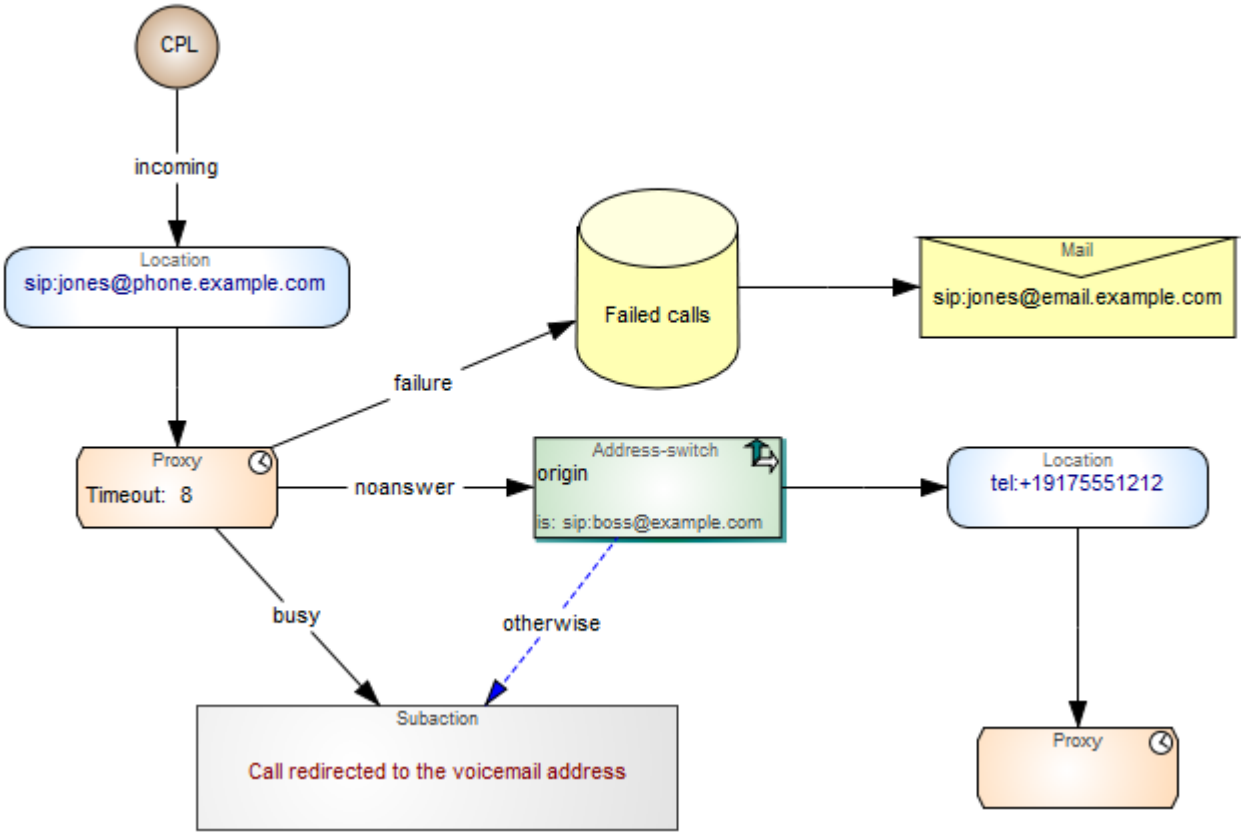

Figure 4. Call redirection reusing voice mail redirection

Fig. 3 および Fig. 4 は、通信事業者の顧客向けの通話処理方法を指定するために使用される CPL 言語での再利用の例を示しています。 Fig. 3 は、ロケーションベースでボイスメールリダイレクト（下部のサブアクションブロック）を使用するコールリダイレクトサービスを仕様として定めています。

同様に、Fig. 4 で定義されているロケーションベースのリダイレクトに関する別の顧客のサービスは、同じボイスメールリダイレクトサービスをより複雑なルールで再利用します。 再利用されるサービスは一度だけ定義され、いくつかのバリアントに適用されます。 次に、バリアントモデルは、再利用されたサブアクションが使用されるコンテキスト（ビジー、無応答など）を詳述します。 つまり、言語はサブアクションを再利用する正しい方法を知っており、仕様定義中にバリアントの作成をガイドおよびチェックできます。

再利用されるサービスは、詳細が表示されるかどうかに応じて、ホワイトボックスまたはブラックボックスコンポーネントとして提供できます。 モデルの再利用を導入すると、他の種類の再利用と同様の問題が発生します。再利用者は詳細を確認したり、変更したり、参照あるいはコピーのいずれによる再利用なのか？ といった疑問です。

また、再利用部品とバリアント固有の部品のさまざまなライフサイクルについて疑問が生じます。再利用部品の更新とバグ修正はユーザーに自動的に配信されますか？　アクセス権、レビューポリシー

などを備えたガバナンスと管理のサポートも、モデリングツールで提供できるでしょう。変動性に対するツールの役割については、セクション 3.4 で後述します。

この小さな例では同じ言語をベースにしています。つまり、再利用可能なサービスは他のすべてのサービスと同じ言語で指定されています。しかしながら異なるユーザーで異なる言語を使用することも可能です（例：オペレーター側のサービス作成用とサービスの実装用 など（Hulshout2007）で紹介されています）。 チーム間で言語が異なる場合、コアチームとバリアント開発チームの責任も明確になります。

3.3 各バリアントでのモデルやモデル要素の再利用をマーク/フィルター/変更する明示的な機能

Approach 2 のように要素を再利用できるようになると、特定のバリアントでそれらの要素をいくぶん変更する必要がしばしば生じます。 たとえば、特定のバリアント（または他の条件）にのみ存在することやアクティブであることをモデル要素にマーク付けしたり、特定のフィルター処理または構成で低レベルモデルを含めるように高レベルモデルを指定できます。 前者はボトムアップのバリアント仕様、後者はトップダウンと見なすことができます。 どちらの場合も、バリエーションを明示的に指定する新しい概念が言語に追加されます（言語は同じままの Approach 2 とは対照的です）。

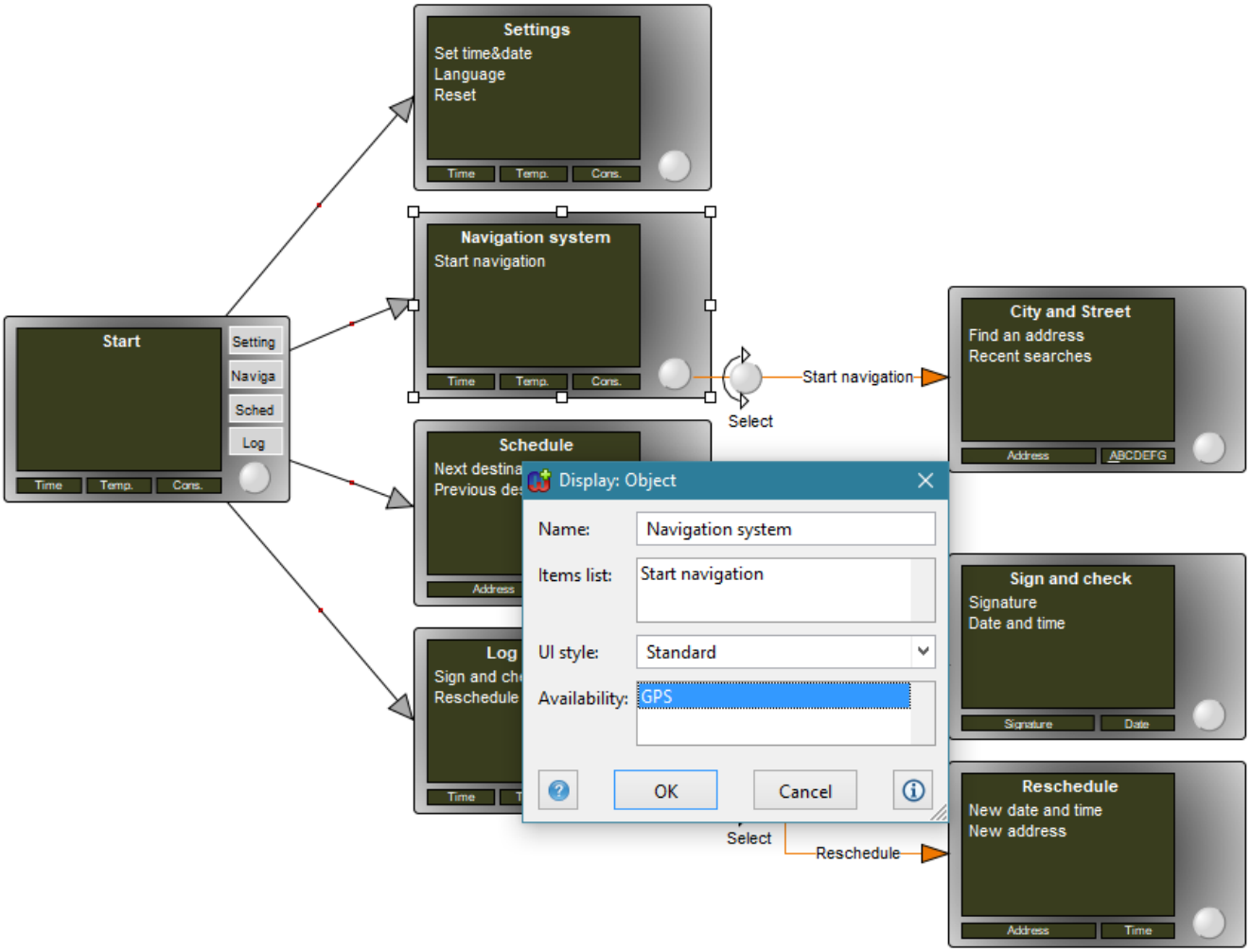

Figure 5. Variation of infotainment system based on sensors available

ボトムアップのバリアント仕様定義例を Fig. 5 に示します。これは、カーインフォテインメントシステムの UI と対話フローの抜粋です。 表示の定義に 'Navigation system' が選択され、そのプロパティダイアログが表示されます。 ダイアログの Availability プロパティ値は、GPS モジュールが利用可能な場合にのみ、この 'Navigation system' 表示が提供されることを定義します。 また、ナビゲーションディスプレイに関連するメニュー、ノブなど他のすべての機能、およびナビゲーションディスプレイを介してのみ到達可能な他のディスプレイも同様です。 変動性は利用可能なセンサーに基づいて設定されますが、製品名（例：インフォテインメントシステム「CarInfo3000」でのみ使用可能）や、機能（例：自動操縦でのみ使用可能）などにも設定できます。このアプローチの弱点は、可用性などのプロパティを入力する必要があることであり、それら複数の場所に変更が起こり得ることです。（この例では、各ディスプレイへの設定）

個々の言語要素にバリアント情報を追加するのではなく、トップダウンアプローチはバリアント情報を新しい上位モデル層に展開します。それ専用の言語を使用して、バリアント用に定義された既存のモデルを構成します。 Fig. 6 は、このトップダウンアプローチを示しており、時計 3 機種（Delicia、

Ace、Sporty）からなる製品ファミリーです。 1 つのモデル図で 3 つのバリアントすべてを指定していますが、各製品に個別の構成モデルが設定されます。 このプロダクトラインの例では、各時計にはディスプレイ（左側）とロジカルな振る舞い（右側）があり、そのサブモデルにはアラームやストップウォッチなどの多くのアプリケーションが含まれています。

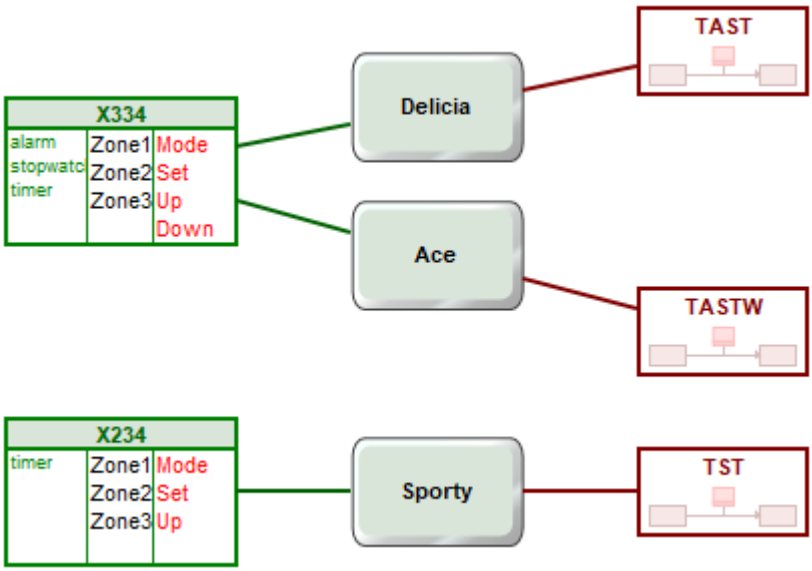

Figure 6. Watch family

ストップウォッチのアプリケーションのモデルを Fig. 7 に示します。ストップウォッチの実行を開始、実行状態で経過時間とアイコンを表示、 ストップウォッチを停止、経過時間の再初期化などの振る舞いです。 3 つのバリアントはすべて、ストップウォッチアプリケーションの同じ仕様を使用しますが、上位のファミリモデルの指定により異なります。 たとえば、機種 Sporty にはタイマーのアイコンがありますが、ストップウォッチのアイコンはありません。そのため、ストップウォッチの実行中は、ストップウォッチアイコンはスポーティーに表示されません。

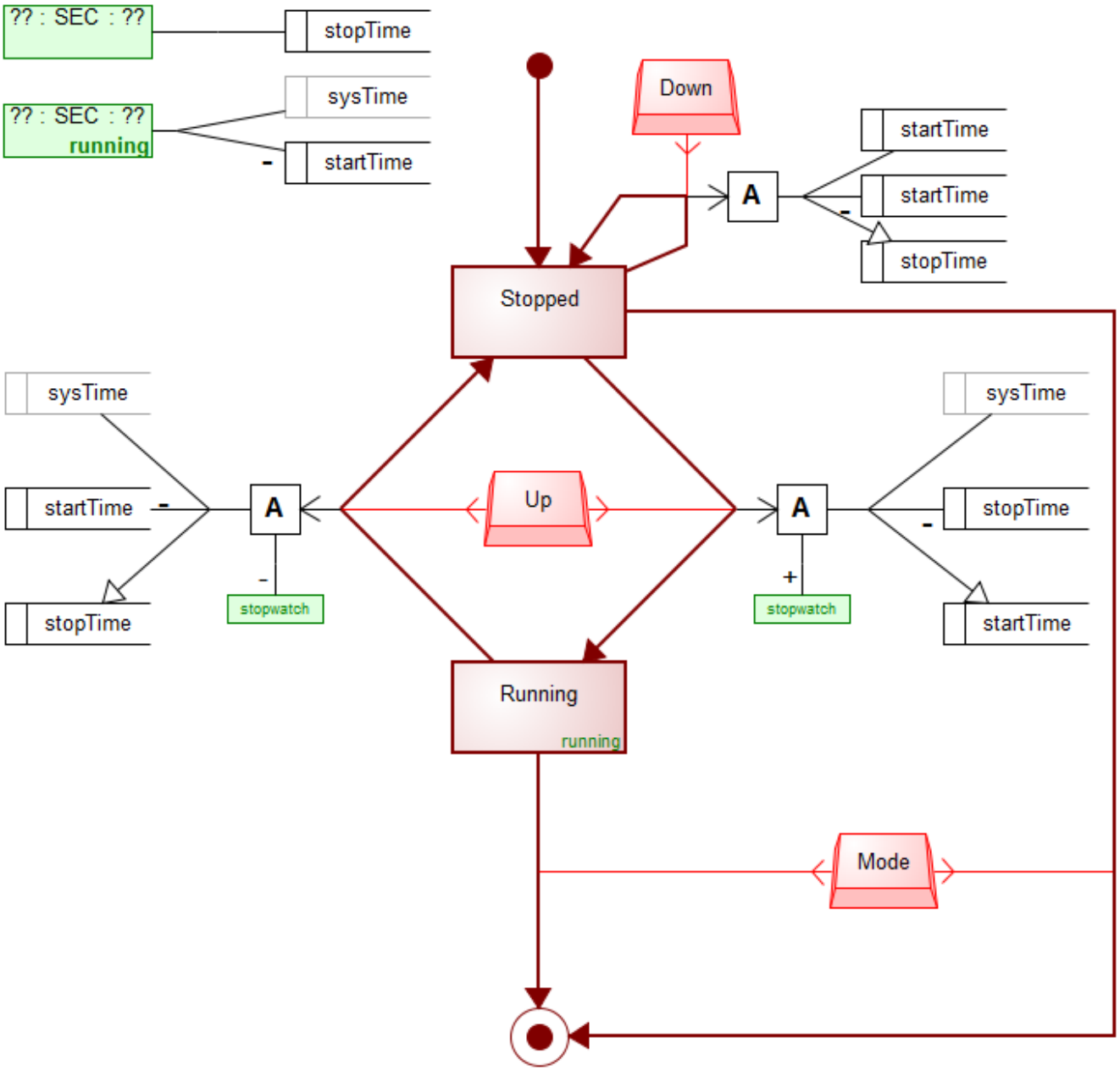

Figure 7. Stopwatch application supporting several variants (Ace, Delicia, Sporty)

この種の言語を使用すると、バリアント開発者は既存のバリアントモデルに必要な機能が既に含まれているかどうかを判断し、それらを使用して構成を追加できます。 したがって、既存の機能を参照および構成することにより、新しい製品を迅速に作成できます。 また開発チームは、既存の製品の機能が維持されるようにしながら、新しいバリアントの既存の機能を拡張できます。

このアプローチに従う言語では、バリアント開発チームはバリアントだけでなく、プロダクトライン全体に対して責任を負います。 作成されたモデルは、バージョン管理と一緒にテストする必要があります。 そしてバリアント開発チームは、モデルがそれぞれ 1 つのバリアントのみに対応する言語を使用する場合とは異なるポリシーを確立する必要があります。

3.4 コアモデルとバリアントモデル

プロダクトライン開発では一般に、「コアチーム」によって開発されたプラットフォームと他の共通性、および「製品開発チーム」によって開発された個々のバリアントを区別します。 DSM は一般にコアとなる共通性をモデリング言語から排除しますが、いくつかのバリアントをモデリングすると、通常バリアントモデルの一部が別のバリアントで再利用され（Approach 2 のように）、後ですべてのバリアントに役立つと認識されます。 このような再利用される「コアモデル」は、新しいバリアントを開始する各チームに提供され、「コアモデラー」専用チームによって個別にメンテナンスされます。 コアモデルのこの明示的な認識、決定、分離は、Approach 4 を Approach 2 と区別するものです。

少なくとも最初は、特定のバリアントのどの部分をコアにするか、どの部分をバリアントごとで指定できるかという明確な分類のためのドメイン自体の知識が十分ではなく、モデリング言語は両方のチームで同じにできます。 言語がバリエーションに対処しない場合は、ツールとプロセスに任されます。 このようなツールベースのアプローチは、ツールの機能と、他のデータと使用される他のツールとの統合に基づいて異なります。

ツールは、コアモデルとバリアントモデルを操作する機能を組み込むことができるでしょう。 例えば、モデルはモデリングツール内の異なるリポジトリまたはプロジェクトで定義できて、異なるチームのアクセスを制限します。 通常、製品開発チームはコアモデルや個々の要素を使用するだけで、変更することはできません。 これに対して MetaEdit +マルチユーザーリポジトリでは、プロジェクトのモデル要素を読み取り専用に設定できるため、コアチームのメンバーのみがそれらを変更できて、他の人はデザインでの参照だけが行えます。 そしてコアが変更されると、バリアント開発チームに更新が自動的に通達されます（モデル図上へのアノテーションなどを介して）。

あるいは、コアモデルが個別にメンテナンスされ、コアチームが専用のコンテンツとバージョンでそれらをリリースすることもできます。 たとえば、異なるバージョンのコアモデルを異なるバリアント開発チームにリリースしたり、コアモデルの異なるセットを異なるバリアントチームにリリースしたりできます。そしてバリアント開発チームは、コアの次のバージョンにいつアップデートするかを決定することができます。

製品開発でそれらを適用するチームには、コアモデルの共有と更新を管理するツールのサポートが必要です。 たとえば、MetaEdit +では、コアチームがモデルをリリースして、製品開発チームが使用する個別のリポジトリにインポートすることができます。また、変更が許可されているかどうかに応じて、異なる権限を設定できます。 コアモデルが変更されると、コアチームは変更されたモデルをリリースし、以前に配信されたコアモデルを自動的に更新することを選択できます。 このアプローチは、すべての設計にアクセスできない下請業者が関与している場合、または開発チームが異なる場所でリモートで作業している場合、あるいは異なる顧客向けに作成されたバリアントモデルを個別に保持する必要がある場合に適用されます（Approach 1 のように）。

3.5　コアの制限されたバリエーション用のコアモデルと言語

分析された事例の中には、再利用されたコアへの変更が一定の制限下で許可されている例もありました。 変更の必要性と許可される変更の種類は、ドメイン分析と言語定義をする中で特定されました。 このアプローチは、ライブラリバリアントの例を用いたクラス図で示されています（Atkinson2002）。 Fig. 8 は、すべてのライブラリシステムに共通する 2 つのクラスのみからなる MetaEdit +で実装された小さなコアモデルを示しています。 このモデルは、コア開発チームによって作成されています。

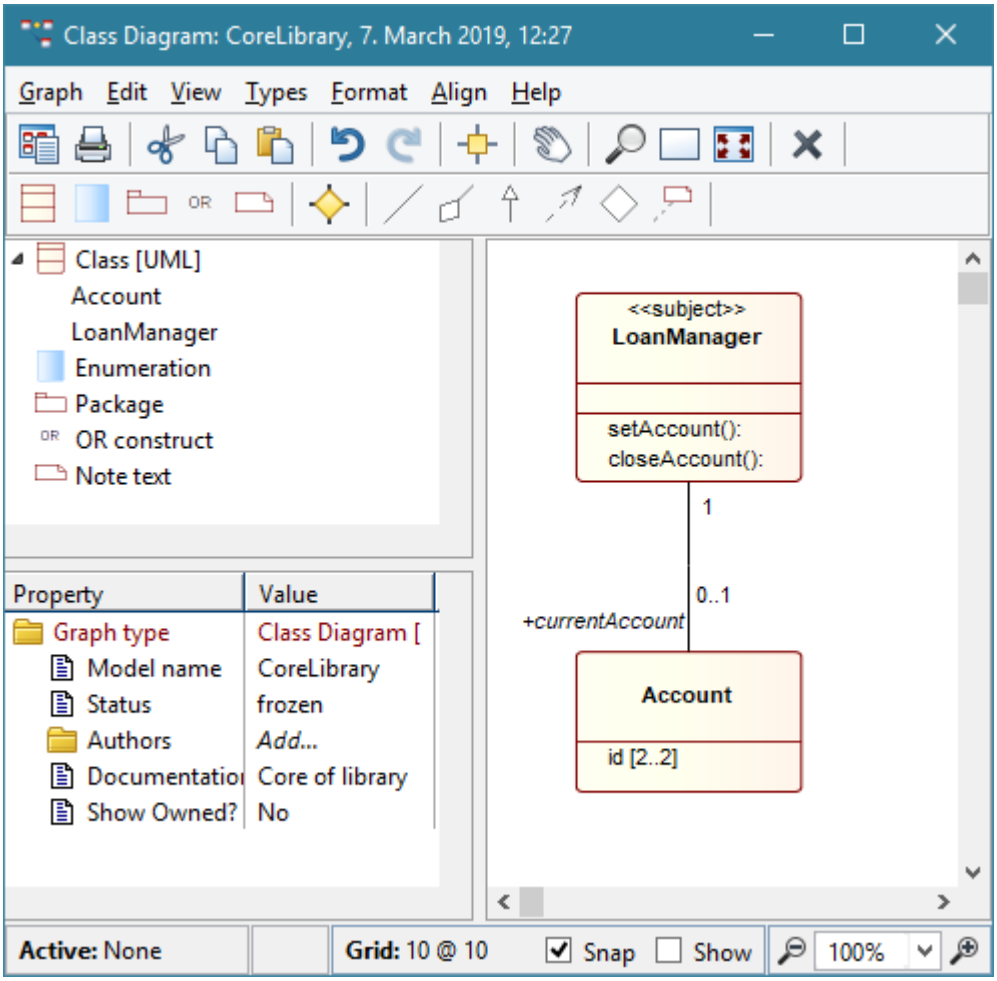

Figure 8. A small example of core model for a library

バリアント開発チームは、コアモデルへの変更を制限されたモデリング言語を使用して製品を作成します。 Fig. 9 は、この言語が使用されるイメージです。 パブリックライブラリのライブラリシステムのバリアントは、コアの“LoanManager”と“Account”の両方を使用しますが、バリアント専用に作成された新しい機能である“ReservationList”を追加できます。 さらに重要なことは、コアの“Account”クラスへの変更がモデリング言語で許可されていて、2 つの方法で変更されていることです。 ローンの最大数の新しい属性が追加され、識別のための既存の属性が必須にされました。 言語

の表記で、コアから使用される部分と、それらに加えられた変更の種類を明確にすることも出来ます。 表記と許可される変更の種類の両方は、言語の作成者によって設定されます。

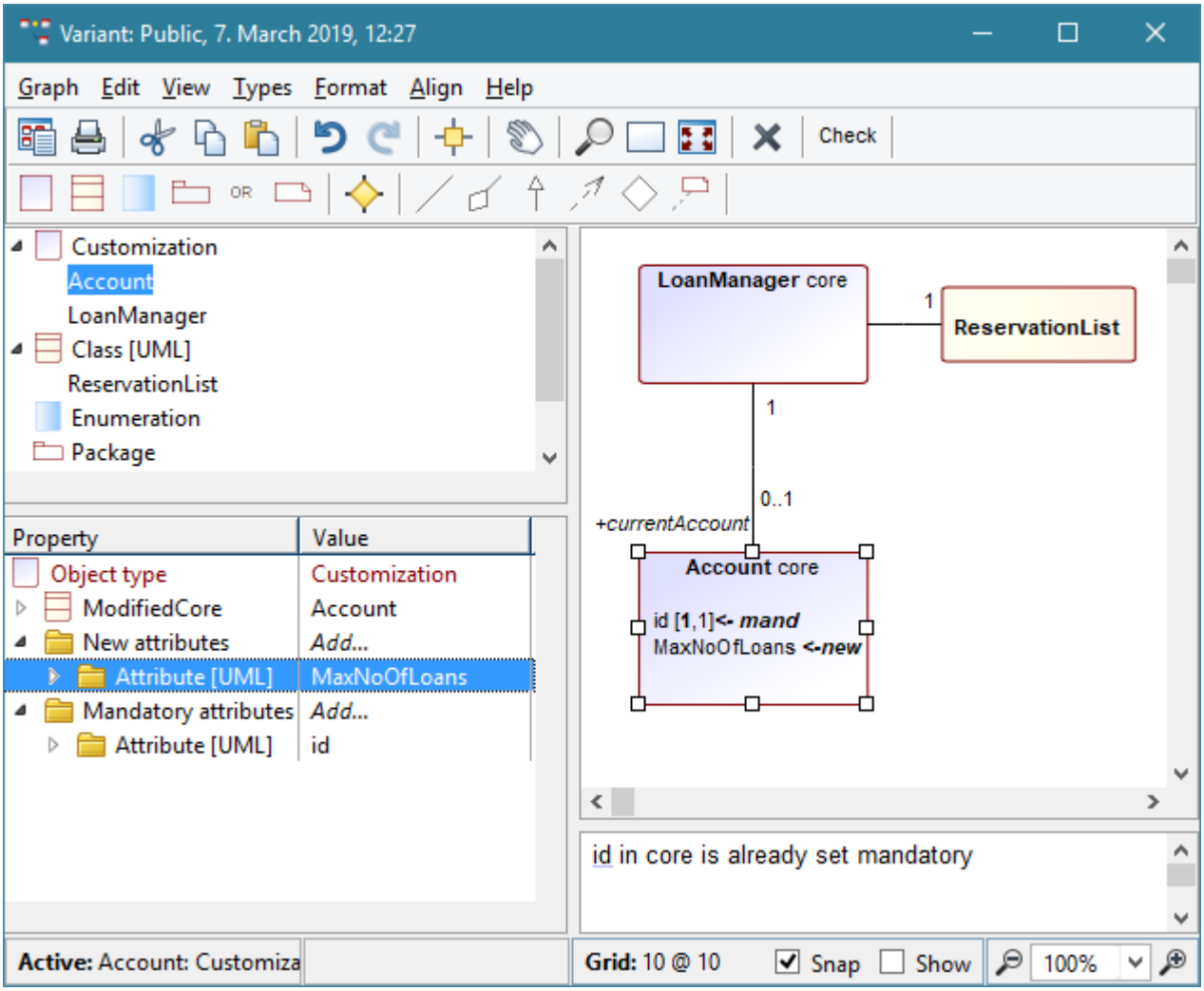

Figure 9. Creating a variant by customizing the core

このアプローチでは、コアパーツを進化させて更新することができますが、同時にバリアントプロジェクトでの修正も可能です。 ツールは、Fig. 9 に示すように、矛盾の可能性をチェックして報告することもできます。ここで、バリアント開発チームは“id”属性を必須にしましたが、後にコアでも必須に変更されました。 MetaEdit +は、バリエーションルールをチェックし、矛盾の可能性についてレポートします。図の右下のエラーテキスト、“id in core is already set mandatory：「コアの ID はすでに必須に設定されています」”の記載を参照してください。 定義された変動性ルールは、モデルの作成時だけでなく、バリアントの編集、生成時にも適用されます。

この例は小さくて、コアを拡張する 3 種のパターンを示しただけのものですが、コアモデルの変更に制限を設定するための同じ原則は、他の種類の拡張ルールや、他のよりドメイン固有の言語にも適用できます。

3.6 マルチレベル：モデル要素は言語要素になる

最後に、ある特定のプロダクトラインの DSM 事例で、マルチレベルモデリングと呼ばれるパターンを特定しました。1 つの役割または特定の開発者によって作成されたモデルを使用して、他の言語要素を作成します。 これは、バリアントに使用可能なセンサーの小さな例を使用して以下に説明します。 Fig. 10 は、可能なセンサーのモデルと、それらが測定するものや適用方法などの特性を示しています。例えば、 継続的なポーリングまたは低エネルギー使用など。 マルチレベルのアプローチでは、モデルはここで使用されるデバイスの機能を指定します。 たとえば、方位用のポーリングセンサーがあり、コンパスの方向を提供します。

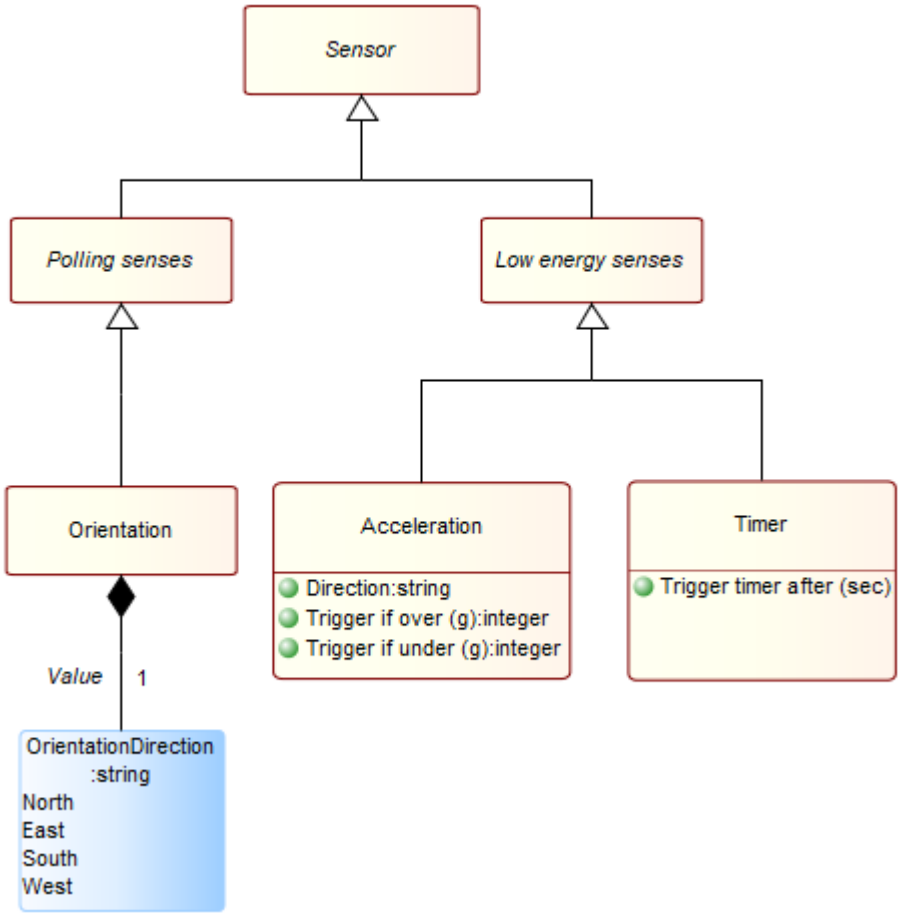

Figure 10. Adding an Orientation sensor language concept

このモデルは、特定のデバイスバリアントのアプリケーションを指定するために使用される言語の一部となります。 Fig. 11 は、方位センサーを使用して追跡を指定するために使用されるこの言語を示しています。 ポーリングベースであるため、このモデルでは、方位センサーからのデータは 15 分ごとに読み取られると記載されています。 コンパスの方向が南の場合、下の遷移がトリガーされ、指定された電話番号にメッセージが送信されます。 利用可能なすべての言語の概念---新しく追加された方位

センサーを含む---は、モデリングツールのツールバーに表示されます。 したがって、Fig. 10 のセンサーを改良すると、Fig. 11 のバリアント開発に使用される言語が更新されます。

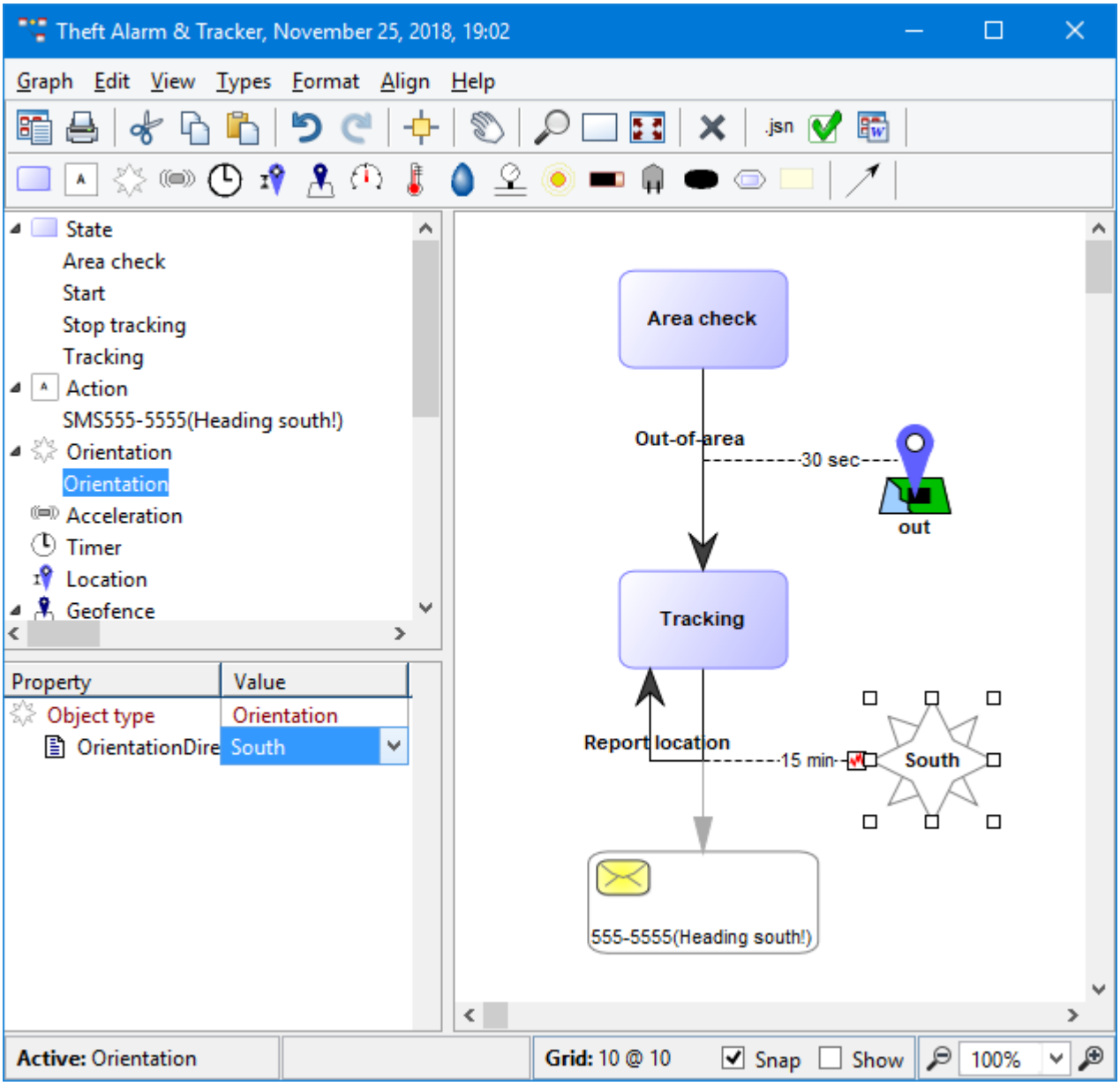

Figure 11. Using an Orientation sensor in a variant

前の 2 つのアプローチのように、さまざまな種類のユーザーまたはロールを持つ可能性があることは明らかです。 言語を変更すると、言語の進化に関する通常のツールサポートの質問も開かれます。幸いなことに、この場合のツールの長所の 1 つですが（Kelly2018）。

4　事例データの評価と分析

分析されたケースのうち、最も一般的なアプローチは、Approach 1：単一バリアント用の言語（9 ケース）および Approach 2：モデルまたはモデル要素をバリアント間で再利用するために設計された言語（7 ケース）でした。多数の言語が単一バリアントに焦点を当てていることは、プロセスが最も単純であるため理解が容易であり、そして、そのような言語を作成するための実証済みの慣行は長い間利用可能でした（例：Weiss1999）。 プロダクトラインエンジニアリングに対するサポートは、言語とジェネレータによって提供され、バリアントの明示的な指定はなく、バリアント間でのモデルの再利用もありません。

取り組みの中で新たな需要が形成されるという我々の経験は、データからもある程度の支持を得ました。 もともと単一のバリアントに焦点を当てていた 2 つのケース（Approach 1）は、後に再利用の必要性に遭遇して Approach 2（\#10, \#11）に移行し、他の 2 つのケースは Approach 3（\#17, \#19）に移行しました。

言語の開発方法を分析したところ、外部コンサルタントの支援なしに社内で開発されたほとんどすべての言語が Approach 1 と 2 に分類されることがわかりました。0.52 の片側ピアソン相関は、プロダクトラインアプローチの明示性の増加と、関連する言語クリエーターの専門知識のカテゴリの増加との間に見つかり、統計的に有意であり、P <0.01 でした。言語の作成者と、完全な本番使用までの道程に沿って言語がどれだけ進歩したかには、相関関係（0.04）は見つかりませんでした。

表 1 から明らかなように、言語のサイズは大きく異なります。最大サイズは最小サイズの 14 倍です。 データはまた、最大の言語の中で長い間適用されてきた言語（\#17, \#18, \#20, \#4）であることも示しました。 0.60 の片側ピアソン相関が、Size と Use の間で見つかり、P <0.005 で統計的に有意でした。Size と Approach の間に相関関係（0.07）は見つかりませんでした。

また、選択したモデリング言語のタイプが Size、Use、Approach に影響を与えるかどうかも調査しました。 言語タイプは、主な焦点または計算モデルに基づいて、場所、コスト、UI、クエリ、構造、ステートマシン、UI フロー、データフロー、アクションフロー、シーケンス、データ構造として分類されました。 多くの場合、1 つのケースに 2 つの言語タイプが含まれていました。 このリストの冒頭で最も単純な 4 つの言語タイプを使用するすべてのケースでは、最も単純な単一バリアント Approach 1 も使用しましたが、これらの言語タイプのサンプルサイズが小さいため、効果は統計的に有意ではありませんでした（これらの言語タイプごとに 1 つまたは 2 つのケースのみ）。

5　結論

この調査では、プロダクトライン開発のためにドメイン固有のモデリング言語が作成された 23 の業界事例を分析しました。 特に、ドメイン固有の言語がバリアントの仕様の再利用をどのように処理するかを調査しました。 事例は、さまざまな業界の色々な種類のプロダクトラインをカバーするように選択されました。 分析に含まれる最も古い言語は 10 年以上使用されており、最新のものは作成されたばかり（2019 年 9 月時点）です。

言語の分析は、その定義（メタモデル）に焦点を当てており、対象領域と作成および採用プロセスに関するより広範な情報も含まれています。 ツールに言語を適用して、バリアント仕様を作成および再利用する方法を調査しました。

調査した事例から、プロダクトライン開発をサポートする DSM の 6 つの異なるアプローチを特定しました。 最も基本的な DSM 言語でさえ、暗黙のうちにプロダクトライン開発をサポートしていました。そして半数以上の言語が、プロダクトラインをよりサポートするように開発されたことは興味深いことでした。言語の 3 分の 1 は、バリアントモデルまたはモデル要素の再利用を追加し、別の 3 分の 1 はさまざまな種類の明示的なバリエーションのモデリングを提供しました。リストは完全とはいえませんが、プロダクトラインをサポートするために適用できる DSM 言語構造の豊富さを示しています。

これらの言語の大部分は、外部コンサルタントのサポートを得ることなく社内で作成されました。 他の言語定義アプローチが適用された場合、外部コンサルタントが関与しました。

これは、これらのケースが言語設計（可変性のための構成の追加など）または特定のツール（マルチレベルモデリングのサポート等）での経験を少なくとも必要としていることを説明します。

各アプローチの利点、作成および保守の労力に関するコスト、および事例内のどの側面が選択されたアプローチにつながったのかを分析するには、より多くの調査・研究が必要です。 将来の作業では、分析するケースの数を増やし、他のツールで作成された DSM ソリューションをカバーすることもできるでしょう。私たちは、他のツールを使用した数多くの業界の事例を調査した研究を知りませんが、他のツールやテクノロジーを使用した言語の作成に取り組むこのような研究を歓迎します。他の研究方法も適用できるかもしれません。例えば、個々の事例内の言語進化のより詳細な分析を提供することです。 逆に、調査によって範囲を広げることができます。

おそらく、この研究の最も有望な側面は、どれだけのケースが MetaEdit +を使用して DSM でプロダクトラインアプローチをうまく実装できたかを見ることでした。未だ多くの組織の開発がクローン＆オウンによって行われているため、プロダクトライン開発の成功を約束できるアプローチとツールは歓迎されることでしょう。

References

(Acher2012) Acher M., Heymans P., Collet P., Quinton C., Lahire P., Merle P., Feature Model Differences. In: Ralyté) J., Franch X., Brinkkemper S., Wrycza S. (eds) Advanced Information Systems Engineering. CAiSE 2012. Lecture Notes in Computer Science, vol 7328. Springer, 2012

(Atkinson2002) Atkinson, C., Bunse, C., Bayer, J., Component-based Product Line Engineering with UML, Pearson Education, 2002

(Cognini2015) Cognini, R, Corradini, F., Polini, A., Re, B., Extending Feature Models to Express Variability in Business Process Models, In proceedings of Advanced Information Systems Engineering Workshops, Springer, 2015

(Czarnecki2000)

Czarnecki, K., Eisenecker, U., Generative Programming, Methods, Tools, and Applications, Addison-Wesley, 2000

(Czarnecki2012) Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wasowski, A., Cool features and tough decisions: a comparison of variability modeling approaches. In Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '12). ACM, 2012

(Hulshout2007) Hulshout, A., Service Creation with MetaEdit+: A telecommunications solution. Presentation at Code Generation Conference, Cambridge, May 19th, 2007

(Kelly1996) Kelly, S., Lyytinen, K., Rossi, M., MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In: Constantopoulos P., Mylopoulos J., Vassiliou Y. (eds) Advanced Information Systems Engineering. CAiSE 1996. Lecture Notes in Computer Science, vol 1080, Springer, 1996

(Kelly2008) Kelly, S., Tolvanen, J.-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley-IEEE Computer Society Press, 2008

(Kelly2009) Kelly, S., Pohjonen, R., Worst Practices for Domain-Specific Modeling, IEEE Software, Vol. 26, 4, 2009

(Kelly2018) Kelly, S., Tolvanen, J.-P., Collaborative creation and versioning of modeling languages with MetaEdit+. In: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '18), Babur, Ö., Strüber, D., Abrahão, S., Burgueño, L., Gogolla, M., Greenyer, J., Kokaly, S., Kolovos, D., Mayerhofer, T., Zahedi, M. (eds.). ACM, pp. 37--41, 2018

(Kouhen2012) El Kouhen, A., Dumoulin, C., Gerard, S., Boulet, P., Evaluation of Modelling Tools Adaptation. CNRS HAL hal-00706701, 2012, http://tinyurl.com/gerard12

(MetaCase2018) MetaCase, MetaEdit+ Workbench 5.5 User's Guide, www.metacase.com, 2018

(Mewes2009) Mewes, K., Domain-specific Modeling of Railway Control Systems with Integrated Verification and Validation, Ph.D. thesis, University of Bremen, 2009

(OMG2017) Object Management Group, Unified Modeling Language, Version 2.5.1, 2017

(Preschern2014) Preschern, C., Kajtazovic, N., Kreiner, C., Evaluation of Domain Modeling Decisions for Two Identical Domain Specific Languages, Lecture Notes on Software Engineering 2, 1, 2014

(Preschern2012) Preschern, C., Leitner, A., Kreiner, C., Domain-Specific Language Architecture for Automation Systems: An Industrial Case Study, In: Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications (eds. Störrle et al.) DTU Informatics, 2012

(Sprinkle2009) Sprinkle, J., Mernik, M., Tolvanen, J-P., Spinellis, D., What Kinds of Nails Need a Domain-Specific Hammer? IEEE Software, Vol. 26, 4, 2009

(Sousa2016) Sousa, G., Rudametkin, W., Duchien, L., Extending feature models with relative cardinalities, Proceedings of the 20th International Systems and Software Product Line Conference, Beijing, China, ACM, 2016, https://doi.org/10.1145/2934466.2934475

(Tolvanen2005) Tolvanen, J.-P., Kelly, S., Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences. Proceedings of the 9th International Software Product Line Conference, Obbink, H., Pohl, K. (eds.). Springer-Verlag, LNCS 3714, 2005

(Tolvanen2018) Tolvanen, J.-P., Kelly, S., Effort Used to Create Domain-Specific Modeling Languages. In ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS 18), ACM, New York, NY, USA, 2018

(Wahyudianto2014) Wahyudianto, Budiardjo, E., Zamzami, E., Feature Modeling and Variability Modeling Syntactic Notation Comparison and Mapping, Journal of Computer and Communications, 2014

(Weiss1999) Weiss, D., Lai, C.T.R., Software Product-line Engineering, Addison Wesley, 1999

(Whittle2014) Whittle, J., Hutchinson, J., Rouncefield, M., The State of Practice in Model-Driven Engineering, IEEE Software, 31, 3, 2014

(Zaid2010) Zaid, L, Kleinermann, F., Troyer, O., Feature Assembly: A New Feature Modeling Technique, Proceedings of 29th International Conference on Conceptual Modeling, Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.). Springer, 2010

FUJI SETSUBI

富士設備工業株式会社 電子機器事業部　www.fuji-setsu.co.jp