Collaborative modeling and metamodeling
コラボレーション開発の技術革新

Juha-Pekka Tolvanen
MetaCase, jpt@metacase.com

Abstract
殆どすべてのソフトウエア開発にコラボレーションが求められるが、モデルベース開発もその例外ではない。そしてコラボレーション開発の技術革新は二つのレベルに分類される。その一つはモデル開発。製品開発者間で共有する仕様を作成して編集し、チェックする中で、開発作業の負担になる衝突への対処や、比較（diff）とマージ作業などを避ける。もう一つはモデリング言語開発。ジェネレータ、モデル表記、ルールや制約、チェック機能等を開発する担当者間にも同様のコラボレーションが望まれる。大規模プロジェクトなら、これら両方の共同作業は欠かすことができない。一人ですべてをまかなえないことは明らかなので。

この資料では、モデルとモデリング言語のコラボレーション開発の技術革新について紹介する。始めにモデリング言語開発（メタモデリング）、次にそれを用いたモデル開発の順で、これらコラボレーションとその恩恵について説明する。そしてコラボレーション開発の技術革新で可能になる、複数のエンジニア、複数のモデリング言語、大規模モデル、モデル変換などへのスケーラビリティについても言及する。

Collaborative language development (metamodeling)

モデリング言語の定義は、抽象構文（言語のコンセプト）、制約、モデル表記、モデル変換やジェネレータにも抽出されるセマンチックなルールなど、多くのパーツからなる。ところが残念なことに、多くの場合、これらパーツの定義は、それぞれ別のドキュメントと作業で実施されている。仕様が分断され、コラボレーションが上手く支援されないことで、矛盾を抱えた不完全な言語に陥りやすい。その例の一つが UML 標準。これは抽象構文に関わる制約（OCL で実装される）に 300 以上の欠陥を抱えていることが報告されている[1]。当然のことながら、言語の全ての仕様が統合されていて、相互にトレース可能な状態であることが望ましい。変更が言語内でトレース可能であり、統合された共通の言語仕様で、担当者同士が作業を行えることなど。

Figure 1 で、これについて ETSI の TDL (テスト記述言語)の開発例を用いて図解する。左上（A）は、エレメント 'ComponentInstance' の、抽象構文の一部を示す。右上（B）は制約の設定例（ユニークな名前を持つこと）。左下（C）は、'ComponentInstance' の具象構文（シンボルなど表記）が、様々なエレメントでどのように定義されているかを示す。そして右下（D）は、TDL のジェネレータ定義画面。ハイライトされている TDL の'ComponentInstance' は、ジェネレータが実行されるとアクセスされる。これ

[1] 361 errors in UML 2.0: Bauerdick et al, in Procs of UML 2004, LNCS 3273, Springer, 2004;
320 errors in UML 2.3: Wilke & Demuth, 2010, journal.ub.tu-berlin.de/eceasst/article/download/669/682

ら 4 つ全てのモデリング言語パーツは上手く統合されるので、何らかの変更を自動反映することやトレースすることができる（例えば抽象構文への変更に対して、制約（B）、表記（C）、ジェネレータ（D）へ）

コラボレーション開発の極端な例では、言語の各パーツの定義が、別々の担当者によって同時に行われる。現実的にはそこまでいかないでも、誰かが抽象構文を定義する傍ら、別の担当者が表記やジェネレータを定義していることは一般に起こり得る。また一つに統合される複数のモデリング言語を開発するなら、それら言語間の統合リンクが定義できる必要があるが、メタモデリング（モデリング言語開発）の革新的なコラボレーションの守備範囲である。

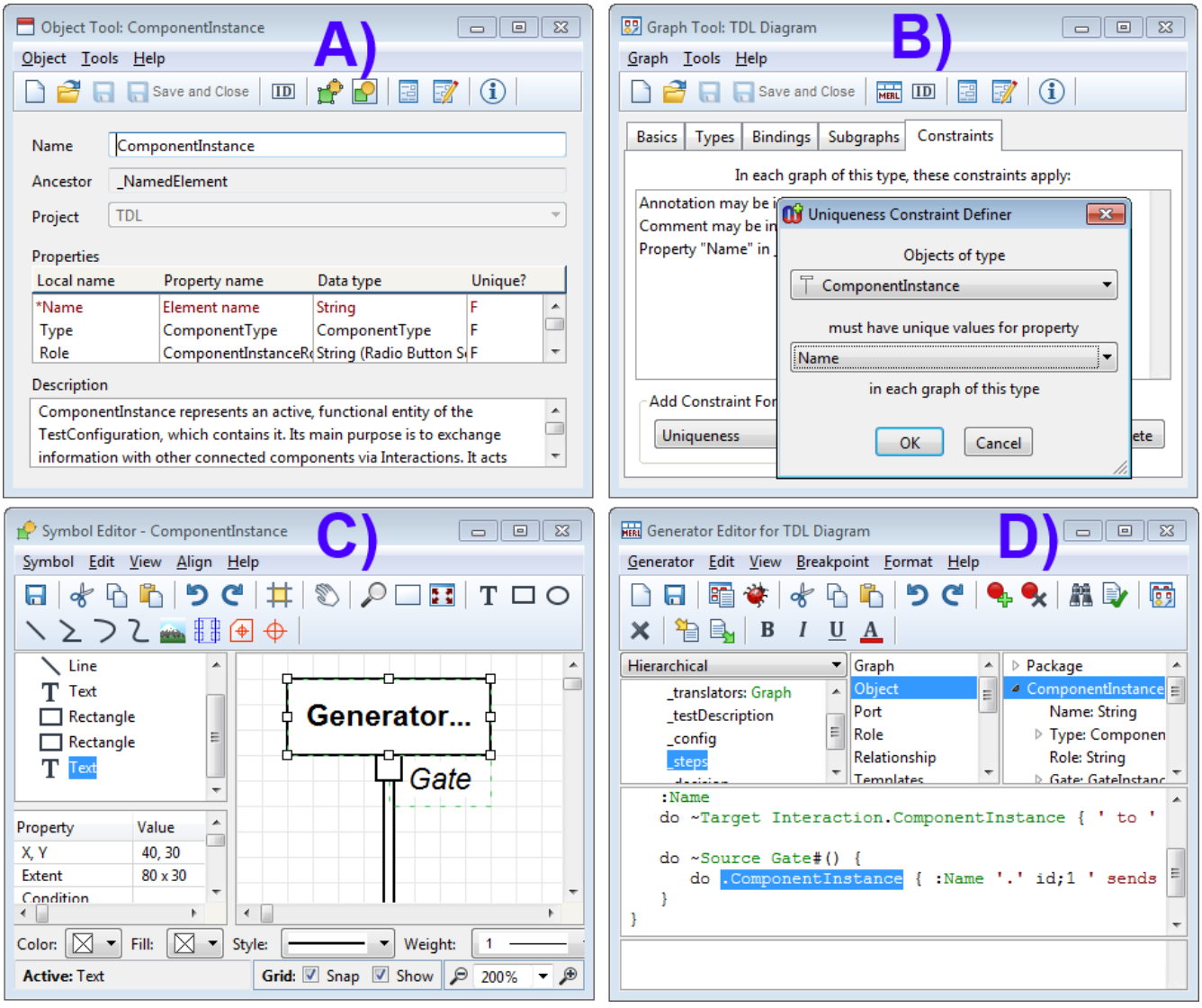

Figure 1. テスト記述言語のコラボレーション開発　A）ComponentInstance エレメントの抽象構文、B）制約やルールの設定、C）ComponentInstance の具象構文（シンボルなど表記）、D）コンポーネントのインスタンスをアクセスする TDL のジェネレータの定義

そして恐らく最も重要なのは、コラボレーション開発には言語開発者間のみに留まらず、そのユーザも同時に関われること。言語開発の共同作業に、そのユーザも関わることで、様々な効果がもたらされる。

- 言語とジェネレータは、早期にそれら定義段階からチェックされる。複数担当者がチームワークで評価して、それらの出来栄えについて議論することや、共同でテストを行うことができる。
- 言語とジェネレータの開発が加速する。複数のジェネレータを個々の担当者で並行して開発できることに留まらず、ジェネレータ開発までにモデル表記やチェックのルールが完全に用意されている必要がないので。
- 早期段階で、言語定義についての評価や妥当性の確認を得ることができる。ユーザによって直ちに評価され、定義の正しさに加えて、ユーザの意図に則した言語であることを確認できる。
- 誤った言語を作ってしまうリスクが軽減され、小さくインクリメントに開発を進めることができる。新規あるいは発展途上のドメインや、未経験者による言語開発の場合にとりわけ重要である。
- ユーザが言語開発の早期段階から関われるので、言語の導入が促進され、支持も得られやすくなる。
- ジェネレータの定義をしている段階であってもモデリングを開始できるので、モデル開発への移行を加速する。特に納期が短くて直ぐにでもモデリング言語が必要な場合など。もしモデリング言語の開発に数か月もかかるなら、肝心な製品開発は終わってしまう。

ここでツールは非常に重要な役割を担うが、多くのツールではモデリング言語の開発に相当な労力を要する[2]。とりわけコラボレーション開発に対する能力が物を言うが、MetaEdit+[3] には共同作業を支援する様々なオプションがある。例えば、言語が定義中であってもユーザが使用できることや、複数の担当者が同じ言語への定義を共同開発できることなど。

Collaborative modeling

モデル開発のコラボレーションもメタモデリング（モデリング言語開発）のそれと同様であるが、製品開発ともなればより多くの担当者が関わって、共有されるエレメントが増えることや、複数のモデリング言語が連携して使用されることになるので、よりスケーラビリティの側面にも話が及ぶ。モデル開発のコラボレーションでは、複数の担当者が同じモデルを同時に編集したい。そして同時に、モデルをフォーマルな状態（単純なお絵描きでは無く）にしておきたいので、アクセス制御をロックする仕組みを、コラボレーションを支援するツールは提供する。そして円滑なコラボレーションには、アクセス制御の粒度を小さくできることが必要である。それにより複数の担当者が同じ図であっても（あるいはマトリクスやツリー構造でも）同時に編集できる。ただ当然のことながら、異なるエレメントに対してのみ。この悲観的ロックは、リポジトリやデータベースで競合を回避する目的に広く採用されている。

---

[2] El Kouhen, A., Dumoulin, C., Gerard, S., Boulet, P., Evaluation of Modeling Tools Adaptation, <hal-00706701v2> 2012, http://hal.archives-ouvertes.fr/docs/00/70/68/41/PDF/Evaluation_of_Modeling_Tools_Adaptation.pdf

[3] http://www.metacase.com

Figure 2 は共同作業で編集されるモデル例（車載システムのアーキテクチャ/システムモデリング言語）で、複数のモデリング言語に対する共同編集作業というスケーラビリティの側面を説明する。

最初に左上 A)では、2 人の担当者が同じモデル図内の異なるハードウエア部品を編集している。そして同時に右上 B)では、システムの論理コンポーネントをアーキテクトが定義していて、左下 C)では、あるエンジニアがコードを生成させるために、A)と B)でまさしく編集されているハードウエアと論理アーキテクチャ間のアロケーションを定義している。そして D)では、機能安全の担当エンジニアが B)で定義される論理コンポーネントのエラーモデルを定義。このようなモデル開発の共同作業から様々な恩恵を受けることができる。

- チームメンバーが同じデザインスペースで同時並行して作業ができる（Fig 2 の例）。フィードバックや意見を求めたい場合は、メンバー全員が同じモデル（あるいはモデルエレメント）を即座に参照して、アップデートすることができる
- 変更があった時に、比較（diff）とマージ作業で衝突を避けるための労力や時間が必要無い
- 必要となる全てのモデル情報を得ることができるので開発が加速される
- 異なるモデリング言語の様々なビューの作業を、モデルレベルに統合できる
- モデルビューやエレメント間でトレース可能。下図 C)の BrakeFL の定義例など
- モデルは早期段階でチェックして検証される。これは単一ダイアグラム内に留まるのではなく、異なる言語を用いて開発される大きなモデルに対しても同じ

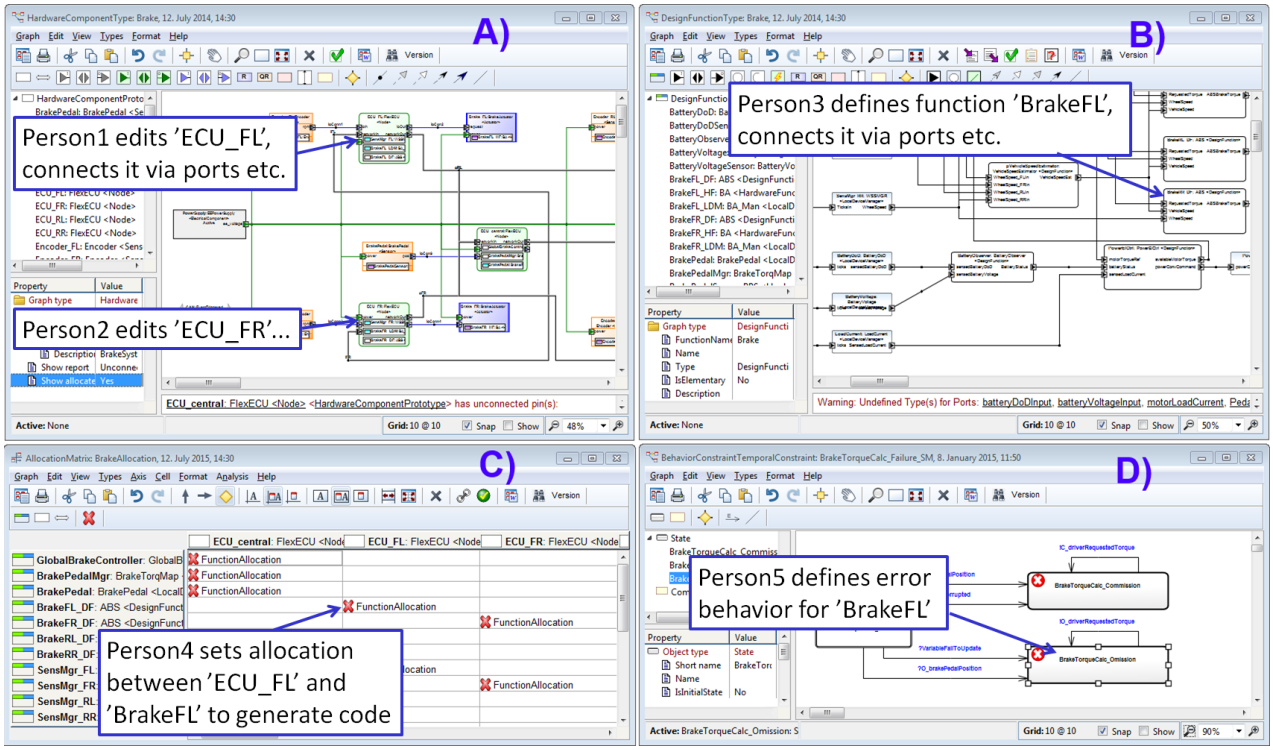

Figure2. 車載システムのモデル開発の共同作業（複数のモデリング言語が統合されたアーキテクチャ/システムモデリング言語） A) ファンクショナルアーキテクチャ, B) ハードウエアアーキテクチャ, C) AUTOSAR コード生成のためのアロケーション, D) 振舞い

モデル開発の共同編集が必要無い場合は（モデルの一部を持ち帰って作業するなど）、一部分をエクスポートして、後からその更新をインポートできる。比較（diff）とマージ作業は意識する必要がない。

Scaling to large

モデル開発に革新的なコラボレーションを支援できるツールなら、産業界の実践的な大規模開発にも順応できる。同じモデルを共同編集できるということだけでは無く、複数のモデリング言語を扱える。そのような複数言語が統合されたドメインスペシフィックな言語は、車載システム開発などで顕著な成功を収めている。Fig 2 のロジカル、フィジカル、ハードウエア、ソフトウエア指向言語等の統合例は、SysML を車載システム向けに拡張した EAST-ADL 言語。単一ファイルごとの編集作業がリポジトリベースに移管することで、モデルエレメント数が何百万に及ぶような大規模モデルへのスケールアップも可能になる。リポジトリが大規模データとトランザクションを扱えることは、金融、テレコム、IT など多くの業界で既に実証済みである。同様に、レイジーローディングでモデルへのアクセスを加速して、トランザクションが発生するときにはモデルデータへのアクセスも容易になる（複数ファイルにアクセスする必要が無いので）。トランザクションが開始する以前に、それらを解析して内部メモリ構造を作る。

規模が増大するアプリケーションの、多様な側面や異なる開発者の様々なビューは、単一のモデリング言語で満たすことはできない。言語・ジェネレータ・表記などの定義を再利用できて、複数のモデリング言語が開発プロセスを通じて統合されて、様々な専門領域が一体化される、革新的なコラボレーションを支援できるツールだけが現実的な選択肢として、モデルベース開発の可能性を大きく飛躍させることができる。