モデルとパターンに基づく静的テスト仕様記述の自動生成

天野 富夫　石川 雄一

Generation of Static Test Specifications Based on Modeling and Patterns

Tomio Amano and Yuhichi Ishikawa

ソフトウェアの品質を向上する上で，上流工程から設計ドキュメントを検証する静的テストは極めて重要である．特に大量の設計ドキュメントが作成される大規模プロジェクトでは静的テストの効率を上げる必要がある．本論文では静的テストにおいて成果物間の整合性を検証する際，検証内容やエラーの原因の記述を自動生成するツールについて報告する．仕様記述の自動化・標準化により，どのような観点で設計ドキュメントを検証し，なぜエラーとなったのかを開発者に正確に伝えることが可能になる．筆者らはUMLを用いて各成果物の構成要素とチェック・ルールの関係をモデル化し，実プロジェクトで必要とされるチェック・ルールの大部分が2つの主要なパターンで表現できることを見いだした．これら2つのパターンに応じて仕様記述を生成するツールを試作し，人手で作成していた仕様記述の問題点を解決した．また，ARC（Analysis & Renovation Catalyst）とGTAM（Guide and Toolkit for Application Modeling）という既存の社内アセットを有効利用することで短期間にツールの実装を行うことができた．

This paper describes a method and a tool which define specifications for static tests of design work products. Static testing is an important activity that assures work product quality, especially in the early stages of development processes. We focused on consistency checking among multiple design documents, modeled work product structures using UML, and found two major patterns that represent frequently used checking rules. Based on these insights, we created a tool for generating check specifications and resolved several problems caused by specifications that were written manually in an ad-hoc manner.

Key Words & Phrases：静的テスト，モデル駆動開発，整合性検証，ARC，GTAM
static test, model-driven development, consistency check, ARC, GTAM

1．はじめに

静的テストとは「テスト対象を動作させずにソース・コードや中間成果物を基に品質を評価するアクティビティー」［1］である．プログラムだけではなく，要件定義や設計の段階で作成されるドキュメントなどに対してもレビューやインスペクションなどの静的テストを行うことでソフトウェア開発の前半（上流工程）で不具合を発見し手戻りを減少させることができる．

静的テストの対象には設計ドキュメントとプログラムなどのシステム・リソースがある．プログラムの静的テストを支援するツールとしてはFindBugs［2］やCheckStyle［3］が広く利用されている．設計ドキュメントの静的テストには意味的なチェックと形式的なチェックがあり，意味的なチェックではレビューやインスペクションの技法が使われている．一方，形式的チェックを効率的に行う具体的な手法・ツールについては十分に体系化されているとはいえない．

形式チェックには成果物間の整合性の検証，トレーサビリティーの検証，定型項目の記述漏れの検証，形式的なあいまい性（網羅的列挙が不完全な「など」といった表現）の検証，重複した記述の検出などがある．筆者らは参加している大規模開発プロジェクト（マクロ設計フェーズの参加者が数百人規模）において，ドキュメントに対する20個以上の静的テスト実施支援ツールを作成したが，本論文ではマクロ設計までの成果物を対象に成果物間の整合性チェック（静的テスト仕様作成支援ツール）について報告する．

実際にコードを動かして行う動的テストと同様，大量の成果物に対して漏れなく効率的に静的テストを実施するためにはツールの活用は欠かせない．成果物のフォーマット標準や成果物間の関連はプロジェクトごとにカスタマイズされるため静的テスト実施支援ツールも適

提出日：2009年8月24日　再提出日：2010年9月7日

宜仕様を決定し作成していく必要があるが，筆者らの参加しているプロジェクトではチェック仕様が不十分であったり，仕様の決定が遅れて適切な時期に実施支援ツールが提供できなかったりという問題が発生した．

筆者らは，成果物の構造をモデル化し頻出するチェックのパターンを抽出することで，成果物間の整合性や追跡可能性に関するチェック・ツールの仕様決定をシステマチックに支援する仕組みとツール（前出のテストの実行自体を支援するツールと区別するため静的テスト仕様作成支援ツールと呼ぶ）を導入し良好な結果を得ることができた．これらのツールはIBM社内で利用可能なアセットを活用して短期間で作成することができた．

以下，2章では成果物間の整合性を検証する手順を概観し，筆者らが経験した問題点を整理する．3章では問題解決のために導入した成果物モデリングと整合性チェックのパターンを説明し，4章でこの考え方に従ってチェック仕様記述を自動生成するツールを実装した例を示す．5章では実プロジェクトの観点での評価を述べ，6章で提案手法の有効性，独自性について考察した後，7章でまとめを行う．

2. 成果物整合性検証の現状と問題点

成果物間で項目の内容が整合していることは成果物間の追跡可能性を担保する上で必須の要件である．CMMI（Capability Maturity Model Integration）［4］のレベル取得においても追跡可能性の重要性が指摘され，RequisitePro［5］などのツールによる管理が推奨されている．しかし，RequisiteProに最初に入力される情報に抜けや誤りがあれば追跡可能性の維持は不可能である．実際にExcel® 形式で管理されていたシステム要求とビジネス・ルールの関係をレビューの後RequisiteProに入力する直前に，システム要求中のビジネス・ルールが一覧に存在するかを検証ツールでチェックしたところ，130個の要求中の40カ所でルールが一覧に存在しないというエラーが検出され修正を行った．人手による確認では抜けや記法の誤りなどをすべて正すことは困難であり，静的テスト実施支援ツールによる整合性チェックは非常に有効といえる．

プロジェクトにおいては1）テスト担当のチーム（以下テスト・チーム）が整合性チェックの仕様を記述し，2）筆者ら開発支援チームがチェックを行うツールを開発後，3）テスト・チームがツールを使ってテストを実施する，4）成果物を作成した開発者はツールの出力結果を受け取った後に問題点を修正する，という手順で作業を行っていた．チェック仕様は日本語の文章記述とそれを補完する図で構成されていた．しかし，特定の成果物の組について整合性が話題になる都度に統一された基準がないままにアドホックにチェック仕様が作成されており以下のような問題点があった．

a）エラーと原因の対応が不明確

1つのエラー番号を出力するチェック仕様を説明する文章中に複数の条件が「OR」や「または」という表現で接続されて記述されていることがあった，そのため開発者は指摘されたエラー番号からその原因を一意に特定することができなかった．

b）文章表現のばらつき

同じようなチェック内容について異なる文章表現で仕様やエラー・メッセージが記述されていた．GD（Global Delivery）メンバーが仕様を記述することもあったため，日本語にして分かりにくい表現も見受けられた．筆者ら開発支援チームが，開発者側が意味を理解できるようにエラー・メッセージを修正することもあった．

c）網羅性に疑問

成果物AとBの間の整合性チェックでは比較している項目を成果物XとYの間では比較していない，といった不整合がしばしば見られた．

3. モデルとパターンに基づく設計

チェック仕様を体系的に定義するために対象となる成果物の構成要素をUML［6］のクラスとしてモデル化し，整合性チェックのルールを構成要素間の関連クラスとして表現した．チェックのルールはさまざまであるが，整合性に関連する主要なものは後述する存在チェックと一致チェックという2つのパターンに集約されると分析し，それらを表現するステレオタイプを導入した．以降の節で，成果物のモデル表現とルール・モデル表現のそれぞれについて説明する．

3.1 成果物モデル表現

開発プロセスを定義する際には成果物間の依存関係を示す物関連図が必ず作成される．特定の成果物間に依存関係があれば，その間で何らかの整合性をチェックする静的テストが必要であることは分かるが，具体的に成果物内のどの項目にどのようなチェックを行うべきかという情報は得られない．

筆者らはそれぞれの成果物内の構造をモデル化し，具体的なチェック・ルールを構成要素間の関連を示すクラスとして表現した．**図1**に典型的な成果物の

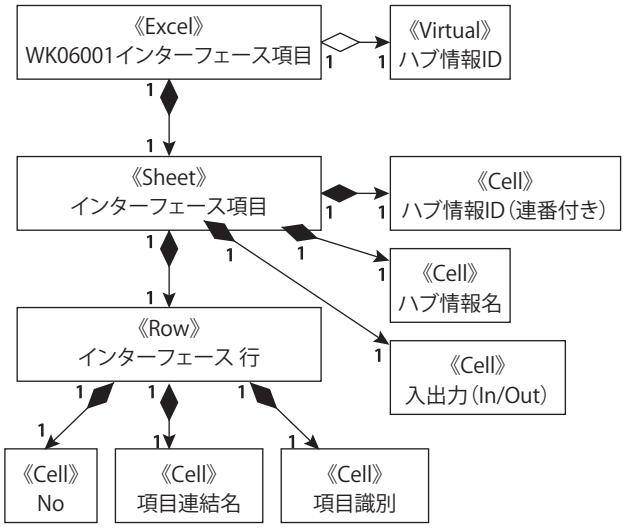

図1. 成果物の構造を示すクラス図の例

構造を示すクラス図を示す．成果物はExcelで記述されているため一般的なExcelファイルの構造を参考に《Excel》，《Sheet》，《Row》，《Cell》，《Virtual》の5つのステレオタイプを定義し各構成要素を分類した．整合性チェックの対象になるのは《Cell》ステレオタイプに属する構成要素のクラスである．これらは，**図1**中の「ハブ情報名」のようにシートごとに記述されている構成要素と「項目連結名」のようにシートを構成する複数の行データ（《Row》ステレオタイプで表現されている）ごとに記述される構成要素とがある．ステレオタイプ《Virtual》は成果物中には直接記述されていないが，記述されていている構成要素から何らかのロジックにより導出され整合性チェックの対象となる情報を表現している．成果物中にはハブ情報IDに枝番の付いた情報が記述されているが，整合性チェックには枝番を取ったハブ情報ID部分を用いるといった場合にこの《Virtual》ステレオタイプを用いて表現する．

3.2 チェック・ルールのモデル表現

ルールを表現するために必要な5つの要素を抽出し，おのおのがほかとどのように関係するかを**表1**のように定義した．「(必須，1)」といった記述はチェックを表すクラスからほかのクラスへの関連（クラス図上での線分）が，必ずかつ1本だけ存在することを規定している．ステレオタイプ《存在チェック》についての記述をクラス図で表現したものを**図2**に示す．

《存在チェック》は整合性チェックの典型的なパターンであり，成果物Aにあったものが成果物Bにもあることをチェックするルールである．《一致チェック》はもう一つの典型的なパターンであり，存在チェックによって特定された構成要素の組について，それぞれに付随するデータが一致することをチェックするルールである．上記2つのパターン以外のルールは《個別チェック》とした．《導出ロジック》は前出の《Virtual》構成要素を生成するためのロジックを示す．「このチェックは項目識別が“個別”または“共通”の場合にのみ行う」といった例外の記述を示すのが《例外条件》である．

表1. ルール構成要素とほかとの関連

存在チェック	• 比較元を示す有向関連（必須, 1） • 比較先を示す有効関連（必須, 1） • 導出ロジックへの依存（オプション, 複数可） • 例外条件への依存（オプション, 複数可）
一致チェック	• 比較対象を示す無向関連（必須, 2） • 前提となる存在チェックへの依存（必須, 1） • 導出ロジックへの依存（オプション, 複数可） • 例外条件への依存（オプション, 複数可）
個別チェック	• チェック対象を示す有向関連（必須1以上） • 前提となる存在チェックへの依存（オプション, 1） • 導出ロジックへの依存（オプション, 複数可） • 例外条件への依存（オプション, 複数可）
導出ロジック	• ほかのチェックからの依存（複数可） • 導出元からの有効関連（必須, 1） • 導出先への有効関連（必須, 1）
例外条件	• ほかのチェックからの依存（複数可）

存在チェック「インターフェース項目中のハブ情報IDに該当する項目がハブ情報一覧中に存在しなければならない」および一致チェック「項目の存在が確認された成果物間でハブ情報名の内容は一致しなければならない」のモデル表現を**図3**に示す．一致チェックから存在チェックへの依存（図中の点線矢印）は成果物の両方にハブ情報IDが存在した場合にのみ，ハブ情報名の一致をチェックすることを示している．

これらの表現に基づいてテスト・チームが文章で記述した仕様（3個の静的テスト実施支援ツール開発依頼）をモデル化したところ9個のチェック・ルールを，7個の存在チェック，3個の一致チェック，1個の個別チェックにより表現することができた．ルールの総数が増えたのは元の文章記述では一つの仕様記述中に複数の条件

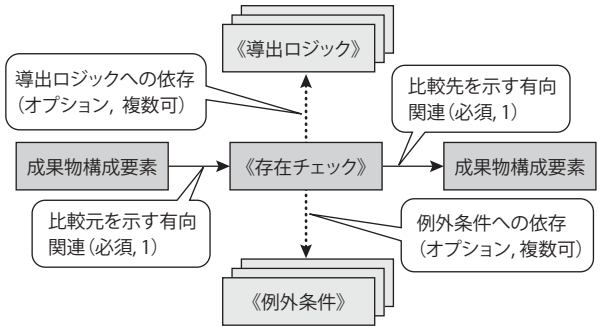

図2. 存在チェックとほかの構成要素との関係

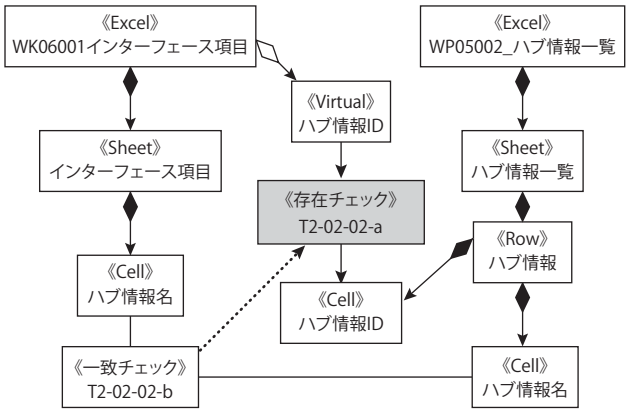

図3. チェック・ルールを示すクラス図の例

が記述されていたのがパターンに従うことで分割されたからである.

テスト・チーム以外にもプロジェクト内のほかのチームからの依頼で作成したツールでも同様の傾向が見られた. コンポーネント処理機能記述に対するプロジェクト標準準拠チェック・ツールのチェック・ルールを（モデリングまでは行っていないが）分析したところ, 184 個のうち 35 個が単独の必須記入項目のチェック, 75 個がそれらの記述フォーマット（「6 桁の数字列か」など）のチェック, 残りの 74 個のチェックはほかの成果物 Excel ファイルや Excel ファイル内のほかのシートの情報との存在チェック, 一致チェックであった. 整合性チェックにおける存在チェックと一致チェックはテスト計画者によらず採用される普遍的なパターンだといえる.

4. ツールを援用した仕様の自動生成

4.1 仕様文章生成方式

3 章で定義したステレオタイプを用いて成果物およびチェック仕様のモデルを作成し, ツールによるチェック仕様文章とエラー・メッセージの自動生成を行った. モデルの作成には弊社の UML モデリング・ツールである RSM（Rational® Software Modeler）[7]を使用している.

チェック・ルールの各構成要素に仕様記述文章のひな形（テンプレート）を定義しておき（**表 2** 参照）, 構成要素間の依存や関連に基づいてこれらのテンプレートを結合して文章を生成する. 存在チェックや一致チェッ

表 2. テンプレート文章の記述例

《存在チェック》の仕様文章テンプレート
Aに含まれる「**B**」と合致する「**C**」が**D**内に存在することを確認する。
《導出ロジック》の仕様文章テンプレート
Aは導出ロジック**B**（**C**）によって**D**から導出する。

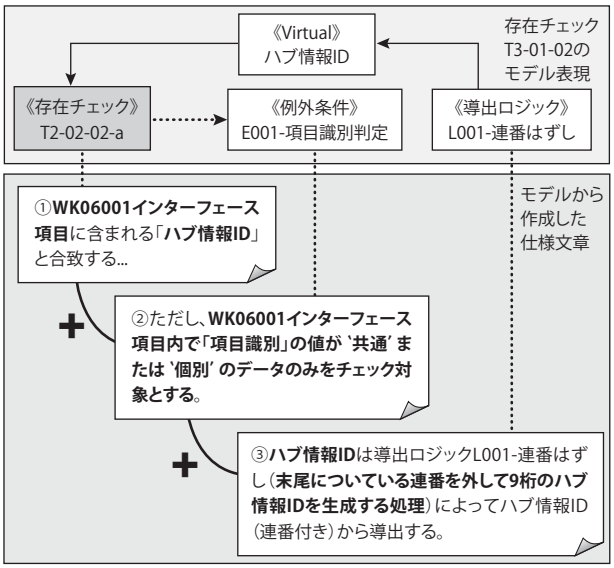

図4. 存在チェックT3-01-02の仕様文書を生成した例
（太字は可変部分）

クのテンプレート中の成果物やその構成要素の名称部分（テンプレート中の太字部分）は可変になっており構成要素間の関連をたどることで値を決定する. 例えば, 前章の**図 3** 中の存在チェック・ルール T3-02-02-a のテンプレート中の可変部分の名称 D は, 1）存在チェックの比較先になる構成要素（この例では《Cell》ハブ情報 ID）をルールから出て行く有効関連から特定し,2）コンポジション関連をたどって最上位要素である成果物名称（「WP05002_ ハブ情報一覧」）を特定することで決定できる.

図 4 は存在チェック・ルールと関連する導出ロジックや例外条件のテンプレートを結合してチェック仕様の文章を生成した例を示す. エラー・メッセージについても仕様文章と同様の方式により生成が可能である.

4.2 社内アセットを活用した実装

前節で説明したような処理を実装するためには, 仕様生成ツールが UML のクラス間の関連や属性の情報を取得する必要がある. 筆者らは GTAM（Guide and Toolkit for Application Modeling）[8] と ARC（Analysis & Renovation Catalyst）[9] という既存の社内アセットを活用することで Java などのプログラミングを行うことなく仕様生成ツールを実装した.

GTAM に含まれる TKAM（ToolKit for Application Modeling）プラグインはモデル要素一覧エクスポート機能を提供しており, チェック・ルールや成果物構成要素の名称や属性, クラス間の関連などをすべて Excel の表形式で出力することができる. 通常, RSM などの

モデリング・ツールの内部データを一括して処理するためには org.eclipse.uml2.uml［10］などの API に習熟した人間がプログラミングを行う必要がある．TKAM が RSM 内部データと Excel 表の相互変換を行ってくれるので，Excel のコマンドやマクロのスキルがあれば UML モデルに対する一括処理を行うことができる．

ARC は，もともと現行システムのプログラムなどのシステム・リソースを分析するためのツールであるが，ドキュメントの静的テストでは，

1）Excel などで書かれた設計ドキュメントを読み込み XML［11］形式に変換する
2）XML ファイルに XSLT［12］を※1 適用し情報の検索や抽出・統合を行う
3）2）の結果を Excel 形式に変換し出力する

という一連の処理を行うフレームワークとして使用している．ツールがルール・チェックや仕様生成を行うためにはドキュメントから必要な情報を抽出しなければならない．ARC ではドキュメント番号を書く場所，各項目とカラムの対応などを XML 形式で定義しておくことで，文書構造に対応した情報を抽出することができる．

図 5 に仕様生成の処理手順を示す．最初に RSM で作成したチェック仕様モデルを GTAM のモデル要素一覧エクスポート機能により Excel ファイルに出力する．前節の仕様文章作成方式に従って文章を生成する XSLT（文章テンプレートの情報をも含んでいる）を用意しておき，この Excel ファイルを ARC で処理することで仕様文章 Excel ファイルを生成することができる．

ARC と GTAM の組み合わせはチェック仕様文章の生成以外にも，UML モデルを操作する上での強力な武器となる．例えば多数の成果物について図 1 に示したような構造モデルを RSM で操作して定義するのは手間がかかる作業である．筆者らは標準チームが作成した各成果物の記述フォーマットを規定する Excel ドキュメントを ARC で処理して，GTAM のモデル要素一覧インポート機能で読み込める形式に変換した．これを RSM に取り込んで編集することで比較的容易に成果物モデルを定義することができた．

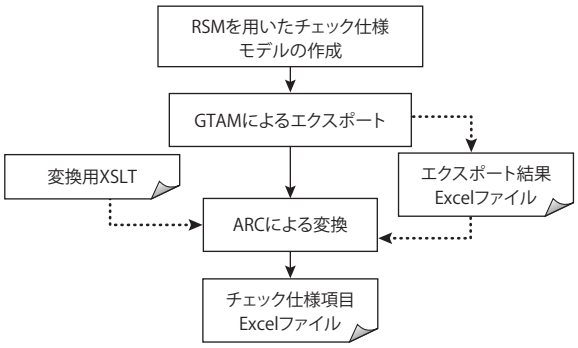

図5. チェック仕様文章の作成手順

5. プロジェクト観点での評価

プロジェクトで実際に行ってきた整合性チェックについて，提案手法によりモデル化を行い，9 個のチェック・ルールを，7 個の存在チェック，3 個の一致チェック，1 個の個別チェックにより表現することができた．これらから，4 章で述べた方式によりチェック仕様の文章とエラー・メッセージを生成できることを確認した．

2 章で述べた現状のチェック仕様の問題点については，本手法により以下のような改善効果が得られたと考えている．

a)「エラーと原因の対応が不明確」について

チェック・ルールの大半を占める存在チェックと一致チェックについてはモデルの書き方が規定されたため，エラーの条件が複数ある場合には条件ごとに別々のチェック・ルールを記述するよう強制されることになった．従ってエラー番号とその原因（エラーと判定される条件）が 1 対 1 に対応することが保証された．

b)「文章表現のばらつき」について

文章テンプレートの導入により，個々の仕様を決めるごとに表現がばらつく可能性は排除された．

c)「網羅性に疑問」について

従来は必要が生じるごとに 2 つの成果物を参照しながらチェック仕様を定義しており，それ以外の成果物の組み合わせでどんなチェックを行っているか，という観点は希薄であった．作成する成果物すべての構造をモデル化しておくことで，例えば「ビジネス・ルール ID はどの成果物に記述されているか，どんなチェックが行われているか」といった俯瞰（ふかん）的なビューを提示することが可能になった（図 6 参照）．このビューにはビジネス・ルールを識別する情報が名前は異なるものの 4 種類の成果物（Excel）の中で記述されていることが表現されている．しかし，ビジネス・ルール ID 間の整合性のチェック仕様が定義されているのはデータ・エンティティー相関図とビジネス・ルール一覧の間だけであることが分かる。俯瞰ビューを用意することで網羅性が保証されるわけではないが，共通の基準に沿ってチェックの必要

※1　XSLT は XML データのフォーマット変換を定義する汎用の言語であり，プログラミングを行わなくても XML データから必要な情報を抽出し所望の形式で出力することができる．

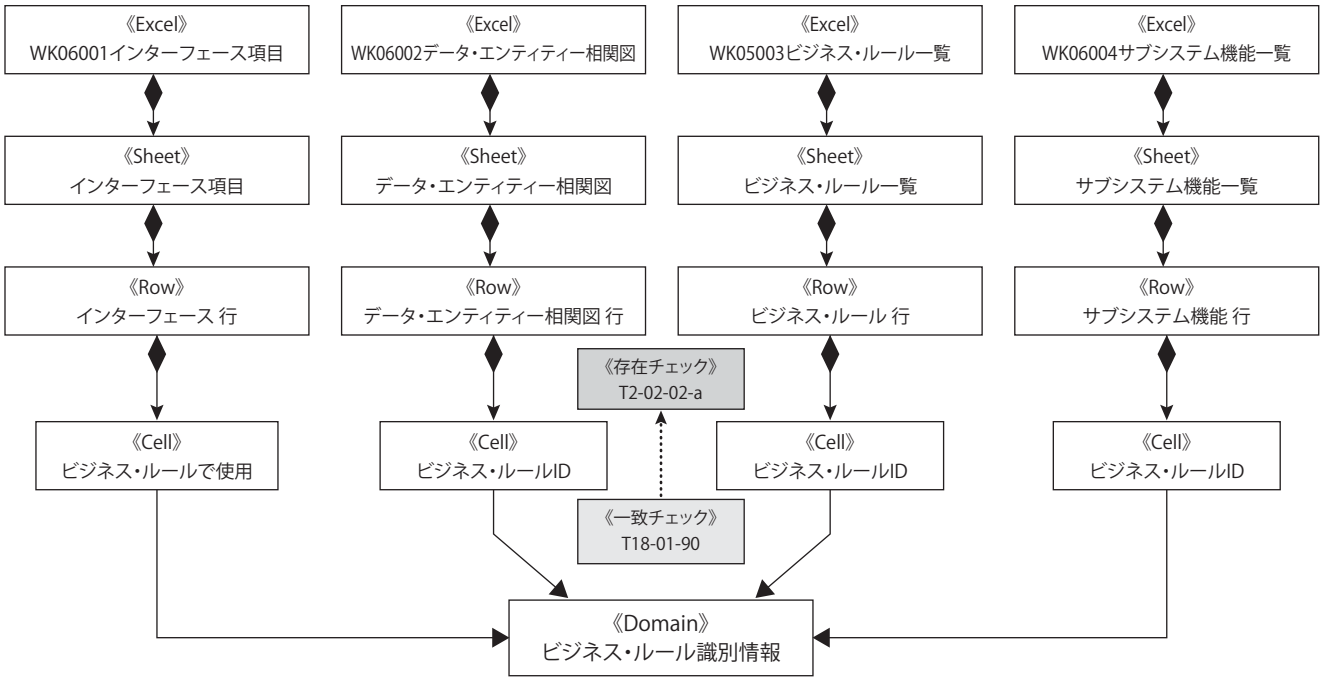

図6. 複数の成果物とチェック・ルール仕様の関係

性を判断する一助にはなると考えている.

6. 考察

成果物単位で依存関係を表現する成果物関連図は多くの開発プロジェクトで作成されているが，筆者らは成果物内部の構造をモデル化した．さらに典型的な2パターンのチェック・ルールを構成要素間の関連を示すクラスとして表現した．これらによって，

1）静的テストの仕様記述文章におけるあいまい性や表現のばらつき，検討漏れを排除できる
2）モデル駆動開発（Model-Driven Development, MDD）の考え方を取り入れて，仕様生成などの作業をツールで支援することができるようになる

という効果を得ることができたと考えている.

1）の効果についてはすでに5章に記述したので，本章では2）について議論する．MDDはソフトウェアの品質や生産性の改善に大きく寄与する一方で，1）支援ツールの存在が必要不可欠であること，2）開発に関わるメンバーにモデリング能力，メタ・モデリング能力などの新たなスキルが要求される，といった点が実現上の障害として指摘されている［13］．上記2つの観点から筆者らが採った実装アプローチをほかの手法と比較し本手法の有効性について考察する.

6.1 支援ツールの実装のしやすさの観点

成果物の構造やチェック・ルールのモデリングはRSMなど既存のツールにより行えるが，チェック仕様文章など固有のアウトプットを生成したい場合には，専用ツールの開発が必要になる．今回，GTAMとARCの組み合わせにより極めて容易に必要なツールを作成することができた．RSM内のモデル情報を取得するためにはAPIをコールするJavaプログラムをコーディングする必要があるが，GTAMのモデル要素一覧エクスポート機能を用いればモデルの情報を一括して出力することができる．最低限，Excelマクロや Visual Basic（VB）の知識があればモデルを操作するツールを作成することができる.

筆者らはさらに，Excelファイルとして出力したモデル情報をARCに取り込みXMLに変換しXSLTを用いて情報の抽出や統合を行っている．表形式になってはいてもUMLモデルの実体はグラフ構造であり，仕様文章の生成に当たっては「《存在チェック》○○が依存している《Cell》構成要素を含む最上位の《Excel》要素は何か?」といった要素間の関連をたどっていく処理が頻繁に行われる．XPATH［14］のパターン・マッチング機能はこのようなグラフ構造上での検索と相性が良く，必要な情報を容易に得ることができる．VBやExcelマクロでも同様の処理をプログラムすることはできるが，XML/XPATHほど直接的には記述できない.

GTAMとARCの利用により存在チェックから仕様の

文書を生成する最初のツールを 2 日ほどで完成することができた．API を使った Java™ プログラムによる実装も検討したが，5 ～ 10 倍の工数がかかると判断し方針を変更した．本論文の例のようにプロジェクト途中で明らかになった問題に MDD を適用する場合には，ツール実装を迅速に行えるスキームが用意されていることが実施判断の重要な因子になるといえる．

6.2 必要スキルの観点

今回，仕様記述を生成する MDD ツールを作成できたのは整合性のチェック・ルールの大部分が 2 つのパターンに集約でき，この 2 つに絞ってツールを作成したことに負う点が大きい．対象を分析し，このようなパターンを抽出するスキルは MDD の実践に当たって必須である．

一方，ツールを実装するために必要なスキルは GTAM と ARC を活用することで大幅にバリヤーを低くすることができた．UML モデルを操作する API では UML の構成要素とは 1 対 1 に対応していない抽象メタ・クラスを扱うため単に UML を記述できる以上のスキルがツール開発者に求められる．UML モデルの入出力用の表現としては XMI［15］を使うこともできるが，この仕様は厳密ではあるが複雑で習熟にかなりの時間を要するため GTAM の出力の代わりに使うのは困難である．

7. まとめ

筆者らはチェック自体を行う静的テスト実施支援ツールも ARC により実装している．今後はチェック仕様から実施ツール用の XSLT ファイルを生成するといった，両者の融合による静的テストの効率化も検討していきたい．

ソフトウェアの巨大化・複雑化に伴い静的テストを厳密にかつ効率的に実施する必要性は増すばかりである．本論文がプロジェクトにおいて静的テストを設計・実施する人々の一助になれば幸いである．

謝辞

静的テストについて議論させていただいている松澤裕史さん，Excel ファイル処理フレームワークを実装していただいた江藤 博明さんに感謝します．

参考文献

［1］ SQuBOK 策定部会編：“ソフトウェア品質知識体系ガイド，” ISBN978-4-274-50162-3 (2007).
［2］ “FindBugs- Find Bugs in Java Programs,” http://findbugs.sourceforge.net/
［3］ “Checkstyle 5.0,” http://checkstyle.sourceforge.net/
［4］ “CMMI 公式 Web サイト ,” http://www.sei.cmu.edu/cmmi/
［5］ “Rational RequisitePro,” http://www.ibm.com/software/jp/rational/products/req/reqpro/index.html
［6］ ランボー他：“UML リファレンスマニュアル,” ISBN4-89471-267-9 (2002).
［7］ “Rational Software Modeler,” http://www.ibm.com/software/jp/rational/products/design/rsm/
［8］ “Rational を活用し生産性向上を実現する IBM の開発メソドロジーのご紹介,” http://www.ibm.com/software/jp/rational/events/rsc2009/pdf/a2.pdf
［9］ “アプリケーション・モダナイゼーション～レガシー to SOA(L2SOA) のご紹介,” http://www.ibm.com/itsolutions/jp/solutions/businessflexibility/events/pdf/impactj_autumn_e-2.pdf
［10］ “Eclipse documentation - Archived Release,” http://help.eclipse.org/help33/topic/org.eclipse.uml2.doc/references/javadoc/overview-summary.html
［11］ “Extensible Markup Language (XML) 1.0,” http://www.w3.org/TR/REC-xml/
［12］ “XSL Transformations (XSLT) Version 1.0,” http://www.w3.org/TR/xslt
［13］ 三ツ井他：“特集モデリングとツールを駆使したこれからのソフトウェア開発技法 ,” 情報処理 45 巻 1 号 pp.1-33 (2004).
［14］ “XML Path Language (XPath) Version 1.0,” http://www.w3.org/TR/xpath
［15］ “MOF 2.0 / XMI Mapping Specification,” v2.1.1, http://www.omg.org/technology/documents/formal/xmi.htm

日本アイ・ビー・エム株式会社
グローバル・ビジネス・サービス /
EA&T ./ SOAソリューション
IT アーキテクト

天野 富夫 Tomio Amano

［プロフィール］
1984 年日本 IBM 入社．東京基礎研究所にて文字認識，文書画像解析，Web サービスの研究・開発に従事．2000 年よりグローバル・ビジネス・サービスに異動，IT アーキテクトとして Web アプリケーションや SOA に関わるシステムの設計・開発を行っている．

日本アイ・ビー・エム株式会社
アプリケーション開発事業
ビジネスアプリケーション・モダナイゼーション部長

石川 雄一 Yuhichi Ishikawa

［プロフィール］
オブジェクト指向技術推進，インターネット普及期の e- ビジネスのシステム開発課長，お客様システム開発でのアプリケーション・アーキテクトなどを担当．現在現行システムの分析，再構築を担当するビジネスアプリケーション・モダナイゼーション部長を担当する IT アーキテクト．