Vol.2011-SE-171 No.24
2011/3/15

統一的表現に向けた仕様記述方法

阿部　睦[†]

仕様記述のために，UML およびその派生言語が利用されることが多い．しかしながら，その利用においてはいろいろな課題もある．本稿では仕様としての表現が求められる内容を網羅でき，実装に対しても設計情報として利用可能な仕様記述方法について報告する．

Specification description method for Integrated Representation

Mutsumi Abe[†]

UML and its derivative language are used for describing specification. However, there are various plobrems using such a language. This paper reports specificarion despription method which cover the requirement as specification and is usable for design information to development.

1. はじめに

仕様の表現において，UML[1]が利用されることが多い．また，特定のドメインに特化した表現のための DSL(Domain Specific Language)として，UML の派生言語である SysML[2],MARTE[3]等も開発されている．しかしながら，仕様記述にあたっては下記に挙げる課題がある．

UML については，

・図の種類が多く，習得に時間がかかる．

・開発物の表現において，単に視点を変えて表現しているだけで，開発物の規模に対して重複が多く冗長となり，読む側の負担が大きくなる．

・仕様の曖昧性の排除のための形式的な仕様表現としての利用が進んでいない[4]

DSL については，

・UML で表現力が不十分な場合，新たな DSL が提案されるが，開発者の習得等，導入へのコストが高い．

・DSL で用意される表現が不十分な場合，さらに新たな表現が追加されてしまう．

・対象領域以外の仕様の表現は他の表現を利用する必要がある．

筆者は，上記のような課題に対応可能で，開発のための仕様を統一的に表現可能な仕様記述方法について開発を行っている．

2. 検討した仕様記述方法

2.1 開発方針

開発中の仕様記述方法（以下，本仕様記述方法）の開発方針では，前章で述べた課題に対応するものとして以下の３点を考慮した．

・少ない構成要素で理解が容易

構成要素を絞ることで習得コストを下げられ，また，表現する際の構成要素の選択に悩む負担の軽減が見込める．

・統一的な表現を可能とし様々な視点からの分析において，表現項目の重複が少ない

統一的な表現とすることにより，表現項目も重複を少なくすることで，可読性の確保が見込める．

・形式的な表現のみで表現し，機械的取扱いが可能

形式的な表現により曖昧性を排除し，また，機械的な取扱いを可能とすることで各種サポートを行うツールの開発が容易となる．

[†] 株式会社トヨタ IT 開発センター
Toyota InfoTechnology Center, Co., Ltd.

さらに上記以外の開発方針としては以下の点についても考慮した．

・機能要件だけでなく非機能要件も表現可能

システム開発においては機能面だけの表現では不十分なため，開発に必要な情報を網羅することが可能な表現が求められる．

・構成要素の組合せを新たな構成要素として追加可能とすることで，記述量の削減による可読性の向上や新たな視点の追加が可能

仕様規模が大きくなると，ダイアグラム中心の仕様記述では記述した仕様の理解・維持管理が容易でなくなるとの指摘もある[5].

本仕様記述方法では，少ない構成要素での表現を目指しているが，少ない構成要素で表現すると，同じようなパターンの記述が頻出し，記述量が増えると可読性が低下する．そのため，構成要素群の接続で同じパターンについては新たな構成要素として追加可能とすることで，記述量の削減が見込める．

また，新たな視点で表現したいといった場合にもその視点に基づいて構成要素を組み合わせることにより創出することができれば，利用者が独自に DSL を定義するのと同じ効果が得られと考える．これは，与えられた DSL を使うのではなく，利用者が表現したいシステムに沿った DSL を定義可能であることも意味し，対象に対する深い理解を促進すると思われる．

・要求表現から詳細仕様まで同一の表現形態とし，工程間での見通しを良くすることで開発効率を向上

工程間で表現の異なる形式を使うと表現内容の翻訳が必要となり，ある工程で意図された仕様が適切に翻訳されない等，翻訳時にバグが入り込む可能性が高くなる．

また，同一の表現形態で且つ形式的な表現であれば，ある用語や対象についての定義も明確になり，利用者間の意識のずれが発生しにくい．

これらの開発方針を踏まえて検討した表現方法の概要について次節以降で述べる．

2.2 表現方法と構成要素

本仕様記述方法はデータを中心として仕様を表現する．データを中心といってもデータ中心アプローチにおけるデータとは異なり，本仕様記述方法における「データ」は機能や振る舞い等も含む広い範囲の物事/事物を対象としている．これは対象を表現する場合，何らかの手段により観測された結果として表現される，つまりデータであるということを意図して構成要素の名称として用いている．

本仕様記述方法の基本的な構成要素はデータと制約の二つである．それぞれの定義としては辞書的な意味では以下の通りである．

データ：状態・条件などを表す数値・文字・記号

制約：物事の成立に必要な条件や規定

データについては，上記で述べた通り，辞書的な意味よりは広い範囲の定義となっている．制約については，辞書的な定義における物事の変わりに本仕様記述方法におけるデータの成立に必要な条件や規定となる．

これらを用いた表現の考え方としては，表現したい対象をまずデータと捉え，それを説明する形でデータに制約を接続する．その際，説明のために必要なデータを制約に接続することができ，そのデータを参照データと呼ぶ．これにより，定義したデータが他のデータと関係付けられる．

接続ルールとしては，データ同士は直接接続されず，制約を常に介する．

データと制約の図形的な表現を以下に示す．

・データは矩形で表現
・制約は角括弧で表現
・データを説明する制約は説明するデータに対して矢印で接続
・参照データは制約に対して通常の線で接続

図 1 は，データと制約の図形的な接続における表現例を示し，変数 A は型で説明され，その際，どのような型かを示すため，int を参照データとして接続している．

図 1　データと制約による表現例

なお，本仕様記述方法では表現法のみを対象としており，表現された内容の矛盾については関知していない．表現内容自体には矛盾があってよく，それらへの対応は必要に応じて別途実施されるべきと考えるからである．

2.3 表現例

本仕様記述方法では，機能や振る舞い等もデータで表現することとしているが，単なるデータ構造以外の表現についての具体的な表現例について以下に例を挙げる．但し，各例はそれぞれの表現の一例であり，本仕様記述表現で規定したものではない．

（1）処理フロー

処理フローにおけるある処理ステップはそのステップの前のステップとの接続で表現することができる．

図２では，「処理２」の前のステップとして「処理１」が定義されている．このとき，対象となるデータが「処理２」であり，その制約が「前処理」であり，参照データが「処理１」と表現されている．

ここで「処理１」から「処理２」に直接矢印で接続されているように見えるが，データ同士は直接接続されないので，「処理２」と「前処理」が矢印で接続され，「処理１」と「前処理」が通常の線で接続されている．これは，矢印と線を明確に分離して接続することも可能えあるが，処理フローということで「処理１」と「処理２」の接続がわかりやすいように配置している．

この図ではまた，ソフトウェア開発に向けた表現として，その処理ステップを実行するための関数名や引数，その処理ステップによる結果などの情報がそれぞれを表す制約とその参照データにより表現している．このように，単に処理フローだけでなく，実際の設計情報も合わせて表現することが可能となっている．
さらに，最大処理時間といった非機能要件についての情報も合わせて表現可能となっている．

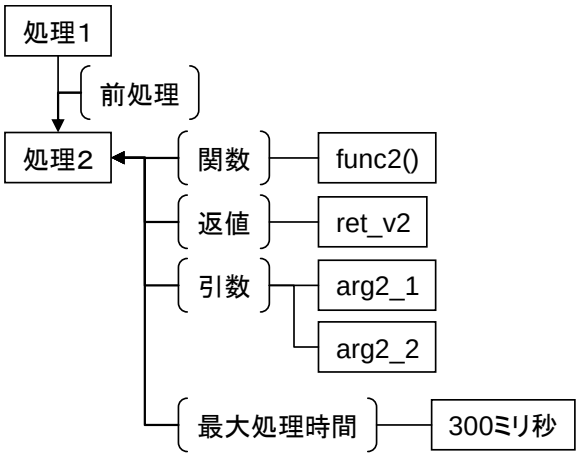

図２　処理フローの表現例

（２）状態遷移

状態遷移は，ある遷移に対して遷移前後の状態と遷移条件を設定することで表現できる．図３では，二つの状態についてスイッチの ON/OFF による遷移を表している．ここでは，遷移の情報に着目して表現を行っており，状態を示す「状態１」「状態２」は遷移を示す「遷移１」「遷移２」の参照データとして表現している．

状態遷移の表現は，状態に着目して表現することも可能であるが，ある状態からもしくはある状態へ遷移する経路は複数考えられるため，遷移に着目して表現したほうが接続した表現が容易だったためである．

ソフトウェア開発に向けた表現では，遷移の条件に判定処理等を具体的に記述していくことで，状態表現と実際の処理とを結びつけることが可能である．

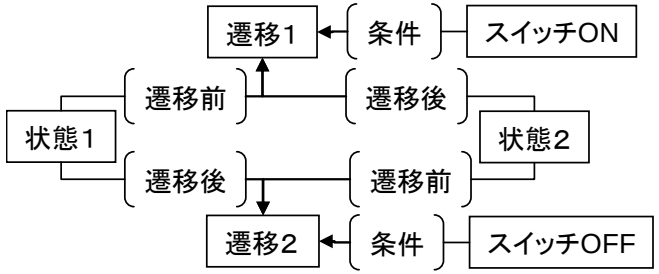

図３　状態遷移の表現例

（３）ユースケース(UML)

図４はユースケースの表現例である．特に UML のユースケースを想定した表現であり，既存の表現を置き換えることも可能となっている．

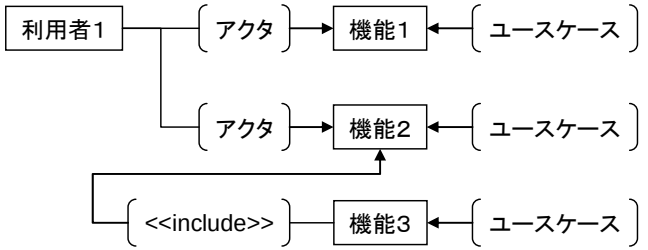

図４　ユースケース(UML)の表現例

このように，単なるデータ以外の物事もデータと制約で表現することが可能となっている．

2.4 類型化

本仕様記述方法では，データと制約の接続パターンを新しい構成要素として追加できる仕組みを用意した．これを類型化と呼ぶ．類型化により，以下の利点が得られる．

データと制約のみの構成要素では，表現対象の規模が大きくなるにつれて，表現量も増大し，可読性が落ちる．そのため，頻出する接続パターンを類型化し，記述量を削減することが可能となる．

下記は，入力から結果を得る「プロセス」という構成要素を追加する場合を示している．

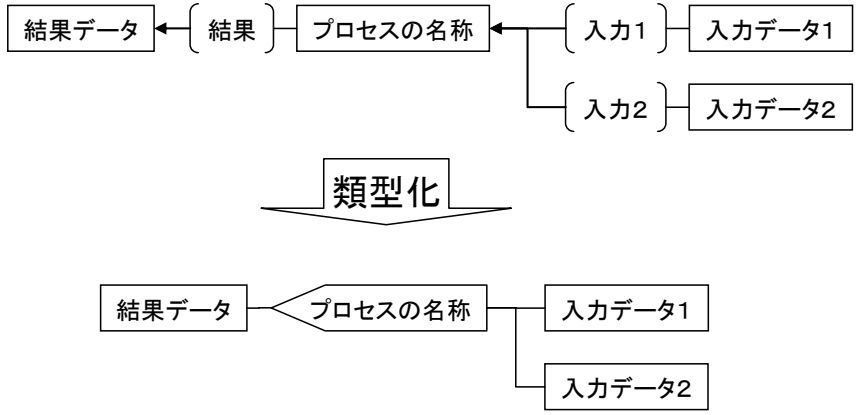

図５　類型化による構成要素の追加例

3. 従来技術との比較

3.1 従来表現との違い

データを中心に考える方法として，データ中心アプローチがある．データ中心アプローチの ER 図では，データを関係で結ぶため，データの関係性は明確である．

本仕様記述方法の場合は上記の通り，個々のデータを説明する形で表現していき，データ間の関連は制約の参照データという形で繋がる．

違いは，本仕様記述方法がある対象に着目し，説明のために他の対象と接続されるという形になっており，対象表現の独立性が高くなっている．例えば，本仕様記述方法では，参照データが無くとも制約のみで対象を表現することも可能である．この場合，他のデータとの関連が無く対象についての完全に独立した記述となる．

3.2 UML との比較

仕様を表現する方法としての比較として，現在最も普及していると思われる UML と具体的な表現についての比較を行う．ここでは，表現方法の違いのみを明確にするため，簡単な例を用いて比較を行う．比較のための簡単な例としては，ある UML ツールに収録されていたエレベータ表現を用いた．

図６は，本仕様記述方法で再表現したものである．

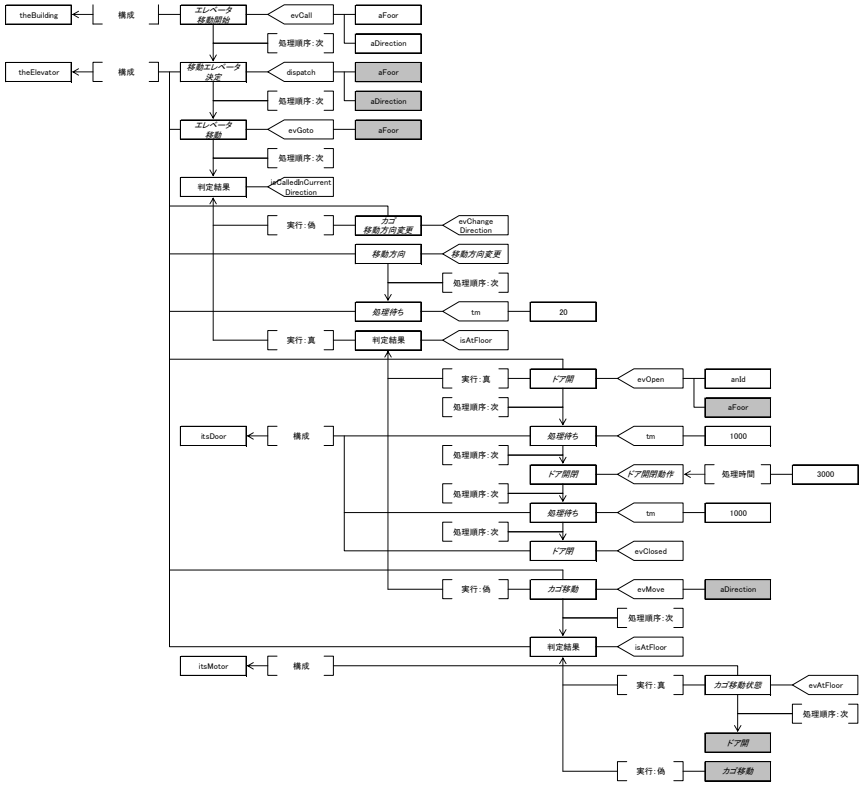

図６　本仕様記述方法での表現

図６の表現の中には，前節で説明したプロセスも構成要素として含んでいる．図中にグレーで表現されているデータは同一のデータを表している．

なお，UML の表現では連携が取れなかった情報については適宜補完して接続し，表現している．

一方，UML 表現では，具体的な図は省略するが，

ユースケース図：１枚
クラス図：１枚
状態遷移図：４枚
シーケンス図：２枚
コラボレーション図：２枚
の計１０枚の図で表現されていた．

このように，UML では複数の図で表現されていたものが，本仕様記述方法では対象を統一的に表現することが可能となっている．

4. 編集ツール

4.1 概要

試作した編集ツールは Eclipse プラグインとして開発し，Eclipse 上で動作する．また，編集結果は XML 形式で保存されるが，データと制約の接続情報と画面配置は個別のファイルとして保存される．

操作はデータや制約などのパーツを配置し，制約を表す線や参照データとの接続を表す線で接続するグラフィカルエディタとなっている．

4.2 操作画面

図７にツールの初期画面を示す．

ツールバーにはアンドゥ/リドゥ，グラフィカルエディタ画面の拡大/縮小，配置図形の各種揃え機能等が配置される．

リソースナビゲータは Eclipse のリソースナビゲータ画面である．

グラフィカルエディタ部が編集のメイン画面である，データや制約を表す図形を自由に配置できる．

プロパティビューはデータの表現名等配置したデータや制約の図形についての情報を編集する画面である．

アウトラインビューは編集内容の縮小して表示する機能である．

サムネイルは選択中の図形等を表示する．

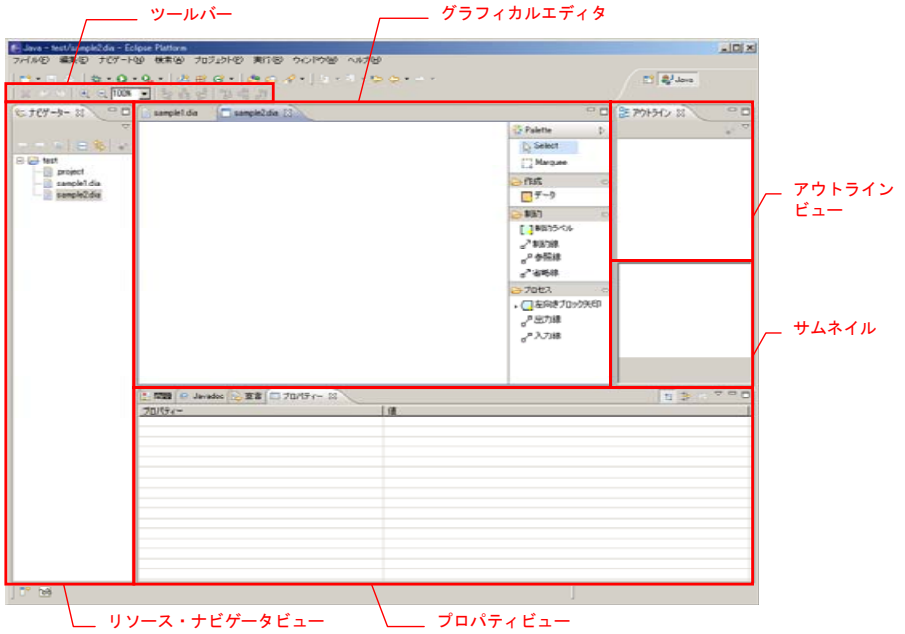

図７　試作した編集ツールの初期画面

5. 考察

第３者による本仕様記述方法による仕様作成による考察を示す．その上で，2.2 節の開発方針への対応について考察する．

5.1 仕様表現作業による考察

考察のため，本仕様記述方法について知識が全く無い開発者数名に，本仕様記述方法の仕様書と試作した編集ツールを渡し，ある題材について実際に表現してもらい，本仕様表現方法についてツールも含めて使用感を述べてもらった．ここでは，それらの使用感に対して考察を行う．

考察にあたっては使用感として共通したものと個別のものについて整理して記載する．なお，題材を用いた仕様の作成は問題なく行えており，これは本仕様記述方法を仕様表現に適用することが可能なことを示している．

5.1.1 共通した使用感

ここでは，開発者に共通した使用感について考察する．

・理解のしやすさ

「データ」と「制約」という二つの構成要素の表現のため，開発者では表現方法の理解には問題は無かった．

これは表記法の理解という面では学習のハードルは低いことを示している．

・表現の自由度の高さ

表現に自由度があるため，最初の段階ではどのように表現してよいかわからないという指摘があった．

これについては，開発者が既に利用している表現方法について本仕様記述方法で再表現して提示する等，対象の表現方法についての例示が必要であることを示している．

・利用者によるルール決めが可能

処理フローの記述や制御構造の表現などで開発者側が独自にルールを決めたり，アイデアを思いついたりとの報告があった．

これは開発者が表現しやすいルールを自ら設定できており，開発者側で利用しやすい形で利用することが可能であることを示している．

・類型化

わかりやすさの向上のため類型化により提供した「プロセス」については使いにくい面も指摘された．

これは，類型化により構成要素の追加する場合は，予め用意する場合は注意が必要であることを示している．

5.1.2 個別の使用感

ここでは，共通の使用感以外の使用感について考察する．

・記述範囲の不明確さ

ある開発者から，今回の記述において，要求分析，基本設計，詳細設計の範囲が明確でなく区切りが難しいということだった．但し，これは本仕様記述方法に限った話ではなく，他の表記でも言えるとのこと．

これは，本仕様記述方法に限らず，工程別に表現する際は表現領域を明確にする必要があることを示している．

・開発領域の知識

ある開発者の考察では実装に必要な仕組みの理解が無いと設計結果が成り立たないという場合もあったとのこと．この開発者からは，UML などの表現手段についての知識や使い方を知っていても，システム設計スキルが無ければ，ただの道具に過ぎないことを改めて感じさせられたとの感想もあった．

これは，開発対象の表現を実装に向けて表現する場合には利用者に実装についての理解が無ければ開発対象の適切な表現ができないことを示している．

5.1.3 ツールについての評価項目

ここでは，ツールについての評価結果を述べる．

・操作的な面

インストールの問題やベンドポイントの操作性，コピー＆ペースト等，基本的な部分としての不十分な面があった．

これは試作した編集ツールがグラフィカルな操作についての能力が不足していたことを示している．

・配置の困難さ

見易さのために配置に工夫したり，同じデータを各所で使用する場合の不便等，配置の困難さに関する指摘がされている．

これは，配置の自由度が高いことで配置することに労力を取られることと，接続関係が複雑になると使いづらいことを示している．

・目的にあった図の必要性

ある開発者の評価では，UML に慣れた人にとっては，ある目的に対応した図が容易されていると表現作業がやりやすいという指摘があった．

これは，対象の表現方法についての例示が必要であることと同義であり，その必要性を示している．

5.2 開発方針への対応

2.1 節で述べた開発方針への対応についての考察を述べる．

・少ない構成要素で理解が容易

構成要素は「データ」と「制約」のみであり理解が容易である．これは，題材の仕様作成によっても明らかとなった．

・統一的な表現を可能とし，様々な視点からの分析において表現項目の重複が少ない

3.2 節の UML との比較において，UML が複数種類の図を用いて表現しているのに対し，本仕様記述表現では 1 種類の図のみで表現できている．

さらに, UML では関連情報の不足のあった部分の発見とその接続が可能であった.

・形式的な表現のみで表現し，機械的取扱いが可能
表現は「データ」と「制約」の接続関係のみで表現され，その接続関係については曖昧性が無く，形式的である．また，編集ツールの内部表現を XML で統一することで機械的取扱いを可能としている．

・機能要件だけでなく非機能要件も表現可能
2.3 節の表現例にもある通り，処理フローのような機能的な表現部分に対して，処理時間といった非機能的な情報を接続して表現できており，単に非機能要件を表現するだけでなく，機能要件の表現と統一的に表現が可能となっている．

・構成要素の組合せを新たな構成要素として追加可能とすることで，記述量の削減による可読性の向上や新たな視点の追加が可能
類型化の導入により，構成要素の組合せによる新しい構成要素の追加が可能となっている．
今回の類型化の実現にあたっては，類型化で新たに定義された構成要素に対して図形表現も合わせて表現することとした．しかしながら，図形表現を合わせて定義することは図形表現作成のコストや図形表現の意味の習得の面で懸念があるため，類型化の実現方法については引き続き検討を要する．

・要求表現から詳細仕様まで同一の表現形態とし，工程間での見通しを良くすることで開発効率を向上
2.3 節の表現例にもある通り，処理フローに対して実際の関数を結びつけることができる等，処理フローを設計する工程と実際の関数を定義していく工程を結びつけることが可能となっている．
また，題材の仕様作成においても要求分析の段階から適用できており，上流工程から実装に近い工程まで表現可能であることは示された．
なお，開発者の使用感で，仕様表現の工程の範囲が明確でなく区切りが難しいとの指摘があったが，本仕様記述方法は同一の表現形態であるため，特に工程の線引きが難しい．ここはガイドラインの整備等運用でカバーするか，ツールで範囲を規定する等の対応が必要であることを示している．

以上のように，2.1 節で述べた開発方針に対して，検討が必要な部分も残っているが，ほぼ対応できたと考える．

6. まとめ

本稿では，少ない構成要素で統一的に仕様を表現可能な仕様記述方法について報告した．本仕様記述方法に基づいた編集ツールを試作し，本仕様記述方法に知識の無い開発者によって本仕様記述方法による仕様作成を行い，仕様記述への利用が可能であることを示すとともに開発方針をほぼ網羅した仕様記述表現が開発できた．
今後は，考察により得られた課題への対応を含め，実際の開発での利用を想定した各種ツールの整備や分析方法，作成方法のガイドラインの整備等を行う予定である．

参考文献

[1] OMG UML, http://www.uml.org/
[2] OMG SysML, http://www.omgsysml.org/
[3] OMG MARTE, http://www.omg.org/omgmarte/
[4] 松村郁生，原口智史，竹内広宜，小野幸一，坂本佳史，中田武男，福岡直明：MDD 移行のための実機計測と設計文書に基づく振舞解析，情報処理学会シンポジウムシリーズ Vol.2009,No.10,pp71-79(2009)
[5] 田中明，高橋修：ビューポイント DSL を用いたシステム仕様記述に関する考察，情報処理学会研究報告，Vol.2010-SE-168 No.13 Vol.2010-EMB-17 No.13(2010)