2007－SE－158 (4)
2007／11／2

SysMLモデルの検証についての考察

広瀬 紳一
日本アイ・ビーエム株式会社 東京基礎研究所

開発プロセスの早い段階で成果物の誤りを発見し，開発コストを削減するための手法として，成果物をフォーマルなモデルとして記述し，そのモデル自身を検証することが有効であると考えられる．様々なタイプのハードウェアやソフトウェアが複合的に組み合わされたシステムのモデリングを目的とした図形的言語であるSysMLを用いて記述されたモデルの検証について，SysMLモデルの持ついくつかの特徴的な側面に着目して見出された検証項目について述べ，それに基づく検証の実現方法について議論する．

On the Verification of SysML Models

Shin-ichi Hirose
Tokyo Research Laboratory, IBM Research

Model verification is a promising technique for detecting errors in specifications and designs at an early stage of the development process. This paper focuses SysML, a graphical modeling language targeting complex systems consisting of various types of hardware and software, and discusses the properties of a model to be verified and the implementation of the verification process based on the characteristics of SysML models.

1. はじめに

従来型の開発手法においては，V字プロセスに沿って最終成果物のテストをすることにより検証がおこなわれていくため，プロジェクトの初期に策定された仕様ほど，後の段階で検証がおこなわれる．したがって，検証時に発見された問題の解決のために，より大きな手戻りのコストが発生してしまうことになる．もちろん，プロセスの各ステップにおいては，レビューという形での検証がおこなわれるが，実際にはそれだけで十分に問題を検出するのは難しい．特に，複合的なシステムの開発においては，最終製品の部品やモジュールへの展開と，それぞれの部品やモジュールの開発・設計という二重の構造が存在するため，問題はより深刻となる．

本報告では，このような問題に対処するためのひとつの方法として，早い段階での成果物に対しても，より厳密な検証をおこなうための手法について考察する．具体的には，SysML[1]を用いたモデリングをおこなうような開発プロセスを対象とし，抽象レベルの高いモデルそのものを検証の対象とすることにより，問題の早期発見を促進することを考える．第2章では，まず，モデル自身を検証することの利点についてさらに考察する．第3章では，今回の対象である，SysMLについて，その元となったUMLとの比較をおこない，続く第4章で，SysMLのモデルに特有な検証項目について議論する．第5章は，実際の検証方法についての検討をおこない，最後の第6章でまとめと今後の課題を述べる．

2. モデルの検証

モデル駆動の開発手法では，モデルの変換に代表される，実行可能コードの自動的な生成の効果が強調されることが多いが，そもそも記述されたモデルが仕様やもともとの目的に合致していなければ，生成されたコードにも

価値はない．したがって，モデリングが仕様を満たすように正確におこなわれていることや，入力として与えられた仕様に誤りがなく，内容が妥当である(そのような製品を開発・製造すること自体に問題がない)ことを確認することも重要である．自然言語で記述された仕様書とくらべれば，たとえ一部であってもUML等のモデリング言語を用いて表現することによって，曖昧さを無くしたり，矛盾を機械的に検出したりすることが可能となるため，全面的にモデル駆動ではない開発においても，モデリングをおこなうことの重要性は広く認知されてきている．

一般に，開発成果物の検証は，その作業の入力として用いられた仕様との突き合わせをおこなうことによってなされる．よく知られた方法としては，たとえば，人手でのレビュー，仕様から導かれる具体的なテストケースを用いたテスト，および，(成果物がモデルの場合に)仕様から導かれるより抽象的な性質をモデル検査等の技術を用いて網羅的に検査する，等のものがあげられる．

形式的手法をうまく適用することができれば，テストケースの質に頼ることなく，モデルの性質が保証されるので最も望ましい方法ではあるが，現状では，扱えるモデルの大きさや複雑性に関する制限が厳しく，他の手法との組み合わせも含め，今後の検討課題であると考える．

以上に基づき，本報告では，成果物としてのモデルに対してテストを実行することによって検証をおこなうことに焦点をあてて議論することにする．さらに，テストによる検証では，テスト方法の設定を工夫することによって，部分的に記述された不完全なモデルに対する検証がおこなえる可能性があり，テスト・ファーストでモデリングをおこなうような手法への発展が期待される．

3. SysMLの概要とその特徴

SysMLは，ハードウェアやソフトウェアを含む複合的なシステムのモデリングをおこなうための図形的な言語である．UMLをベースとして，システム・エンジニアリングのためのプロファイルを付加し，また，同時にシステムの記述には不要であると考えられる要素を使用しないものとして取り除く形で定義されている．もちろん，プロファイルによる拡張であるため，他のプロファイルとの併用も可能である．

以下の節では，特に，開発プロセスの初期段階において，実装レベルの成果物がない状態で，SysMLモデル自体の検証をおこなう場合に必要となる拡張部分について簡単に述べる．様々なタイプの対象をモデリングするために導入された，連続的なシステムを記述するための構文などもSysMLの大きな特徴ではあるが，後述する理由により，本報告では割愛されている．SysMLの全体像をはじめ，その背景や歴史については，[2]のサイトの文書等を参照されたい．

3.1 ブロックとブロック定義図

SysMLでのモデリングにおける基本となるモデル要素はブロック(block)と呼ばれる，UMLのクラスのサブクラスである．UMLのクラス図では，クラスの間の様々な関係を表現するが，SysMLのブロック定義図(Block Definition Diagram, BDD)は，それに加えて，モデリングの対象となるシステムがどのような構造を持っているかを記述することが，その主たる目的となる．したがって，最も重要なブロック間の関連は，コンポジションの関連であり，これで結ばれたモデリング対象の構造を表現しているブロックの木構造がブロック定義図の核となる．

図1に簡単なブロック定義図を示す．ここで注目すべき性質として，コンポジションの関連で結ばれた親のブロックは，それを構成する部品である子ブロックを組み合わせたものに等しい，という点があげられる．これは，物理的な実体を持つ対象を例に考えるとわかりやすいであろう．たとえば，ベアリングが外輪，内輪，リテーナーといくつかのボールからなるとすれば，製品であるベアリングから，全ての部品を取り去ってしまうと何も残らない，ということである．

3.2 内部ブロック図

内部ブロック図(Internal Block Diagram, IBD)は，UMLの複合構造図を拡張したものであり，UMLの場合と同じく，ブロックの構成要素間の関係を記述するために用いられる．部品の中をさらに展開して表示することができるのもUMLと同様であり，多重の入れ子になったブロックについても同じ図の上での関連づけが可能である．連続的なシステムの記

述のための拡張がSysMLでの主な変更点であるため，本報告の議論においては，複合構造図とほぼ同じものと見なすことができる．

図2は，図1のブロック定義図に対応する内部ブロック図の例である．

3.3 制約ブロックとパラメトリック図

SysMLの大きな特徴として，ブロックの属性間の関係を明示的にモデリングするための手段が用意されていることがあげられる．

制約ブロックは，対象に依存しない形での制約(式)を表現するための特殊なブロックである．図3に制約ブロックの例を示す．このような知識を独立したブロックとすることかもわかるように，制約ブロック自体は，ひとつのモデル特有な制約を記述するものではなく，ある分野での知識を表現するための要素としてとらえられており，制約ブロックからなるモデル・ライブラリーを定義し，それを自分のモデルに組み込んで参照するといった使い方が提案されている．

一方，パラメトリック図(Parametric Diagram)は，開発対象のモデルにおいて，制約ブロックに定義された式を用いて表現できるような属性間の制約を具体化して表現するためのものであって，内部ブロック図の，制約条件を表現するための特殊な拡張として定義されている．ブロックの属性間の制約を表わすためには，制約を課すコンテキストとなるブロック(コンポジションの関連によって，その子ブロックもすべてコンテキストに含まれることになる)と，適用される複数の制約ブロックを子に持つような，特に具体物には対応しないブロックを定義し，そのブロックに対するパラメトリック図を作成する．

図4は，制約のコンテキストとなるブロックのブロック定義図の例，図5は，そこで定義されたパラメトリック図の例である．制約ブロックは，他の要素とは異なり，角が丸くなった長方形で表現されており，自身のパラメーター(小さな正方形で図示)と通常のブロックの属性とを関連付けることにより制約式を具体化する．

SysML自身は，制約式を表現するための言語を指定していないので，制約の解釈は，最終的にそれが成立することが求められていることの言明にとどまる．したがって，ある時点での制約違反の解消を，実際にはそれが可能であるとしても，どのようにおこなうかの規定もなく，制約を満たすために適当なブロックの属性値を変更する場合の指針となるように，他の属性の値に従属的な属性の定義をおこなうといったこともできない．そのため，パラメトリック図を検証に利用するためには，利用する制約ソルバーの動作に関する理解が不可欠となるという課題がある．

3.4 要求図

SysMLを特徴づける大きな要素として，要求(requirement)を直接的にモデル要素として扱うことがあげられる．要求図(Requirement Diagram)は，基本的には，モデリング対象のシステムに対する要求群を，基本的な要求とそこから要求分析等をおこなって導かれた(deriveReqtで関連付けられる)ものとの関連も含めて表現するものである．また，ある要求がどのブロックの実現によって満たされるかを表現するための関連(satisfy)が定義されている．

要求の内容自体は，自然言語で表現されていることを前提としているため，その曖昧さを補うために，ユースケースや振る舞い(behavior)を表現する要素にUMLのrefineを用いて関連付けることが可能になっている．同様に，その要求が満たされていることを検証するための振る舞いにステレオタイプ≪testCase≫を付加し，それと要求とをSysMLで定義する関連verifyを用いて結びつけることによって，要求図の中によりフォーマルな仕様や，検証の手順を記述することができる．

このような様々な関連付けによって，特定の要求に注目して，それを実現するための機構やテスト項目を取り出すことが可能になる．

図6は，要求間の関係を表現した要求図の例であり，図7は，要求と他のモデル要素との関係が表現された要求図の例である．

4. SysMLモデルの検証

ここでは，以上のようなSysMLの特徴に基づき，モデル自体の検証が有用であると考えられる局面について考察する．

一般には，SysMLを用いてモデリングをおこなうような対象は，離散的なイベントに基づいて制御をおこなうコントローラーと連続的に動作するセンサーやアクチュエーターからなる

ハイブリッド・システムであると考えられる．とはいえ，時間的・空間的に連続的な動作をUMLの振る舞いとして簡明に表現するのは簡単ではない．SysMLでは，いくつかの拡張によって，表現できる範囲は拡大しているものの，連続システムの記述やシミュレーションにより特化した既存の処理系が存在することを考えると，全てを自身の中でおこなうのではなく，それらとの連携を考えていくべきであろう．

ここでは，SysMLモデルのみで十分に表現することができると考えられる，離散的なシステム，もしくは，離散的な近似として表現されたシステムのモデリングを対象に議論をおこなう．

4.1 単純なケース：ブロック仕様の検証

前述のように，SysMLでは，自然言語での仕様記述を補うことを目的として，様々な要素に振る舞いを付加することができる．したがって，制御機能を持つブロックでは，状態機械を用いて，主たる機能を表現することが自然である．この時点で，イベントが同定できていることになるので，仕様書に記述された動作を，状態機械への入力イベント列とそれらのイベントに対する状態機械の反応として書き直せば，それをテストケースとして，状態機械の振る舞いを検証することができる．また，製品として，たとえ仕様記述がなくても当然持っているべき機能（たとえば，エラーが起きた場合の振る舞い等）を，製品によらない，より一般的なテストケースとしてライブラリー化しておけば，それらの検証をおこなうことで，仕様自体の妥当性の検証に応用することも可能であろう．

このようにして，イベント（列）に対する反応を検証することができるわけであるが，言うまでもなく，状態の遷移を追跡するだけでは，非常におおまかな振る舞いの検証にしかならならず，開発が進んでも期待される動作が得られるということにはならないのは明らかである．そこで，さらに，状態機械の遷移や状態に付随する振る舞いをアクティビティーとしてモデリングすることによって，対象システムの振る舞いをより詳細に表現することを考える．そのために，ここでは，ブロックに外部から観察可能なプロパティーとしてのUML属性を定義し，状態機械が持つアクティビティーはそれらの属性を更新するように振る舞うものとしてモデリングをおこなうこととする．これは，モデル駆動開発において，アクティビティーを最終的なコードにつながるものとして記述するのとは多少異なり，あくまで，現在のモデリングのレベルで観察できる性質の表現であるとするのが良い．

また，検証の実行に際して，対象のブロックに関連するパラメトリック図を検証の対象に加えることも考えられる．パラメトリック図は，本来は，複数のブロックの属性間の制約を表現するためのものであるが，ひとつのブロックだけを考える場合でも，たとえば，ある属性の値が指定された範囲内にある，といった制約を表現することも可能ではある．ただし，このような制約は，もともとUMLで要素の制約として表現が可能であるので，実際にどのような制約をどの方式で表現するのかはモデリングの対象に応じて開発プロジェクトごとに判断していくべきである．

4.2 親ブロックの子ブロックへの分解の検証

あるブロックの制御システムとしての仕様が検証され，かつ，それが単一の部品で実現されるものではないと判断された場合には，そのブロックをいくつかの部品に分解し，ブロック定義図や内部ブロック図を用いて，どの部品をいくつ，どのように組み合わせて目的を達成するのかを表現するモデルをつくることになる．

この時，組み合わされた部品が，親ブロックで検証されたものと全く同じ振る舞いをすることが必要であることは，前述のとおりである．これを言い替えれば，すべてのブロックの分解に際して，各子ブロックの振る舞いを記述する状態機械の集合が，親ブロックの仕様である状態機械と同じ振る舞いをすることを確かめることによって，ブロックの分解の正しさを検証できるということである．

この目的での検証に用いるテストケースは，親ブロックの状態機械を検証するために用いたものを再利用することができる．また，前節と同様に，各子ブロックに，状態機械の持つアクティビティーで参照・更新される属性を定義することによって，より詳細なレベルでの検証をおこなうことができる．

さらに，検証の対象となるブロックとそれに付随する状態機械が複数あるので，パラメトリッ

ク図で記述されるプロパティー間の制約は本来の意味を持つものとして利用することができる. このような制約条件は, 一般に, ブロックの属性が更新されたときに, 必要に応じて評価されることになる. ただし, 単に制約違反がレポートされるのか, できるだけ制約条件を満たすように, 他の属性値を調整するなどの変更の伝播がおこなわれるのかは, 制約ソルバーの機能に応じて変わりうるので, アクティビティーの記述に考慮が必要となる.

5. 検証の実行手法

本章では, 前章での議論に基づき, ブロックの主たる振る舞いを表現する状態機械を, テストケースとしてのアクティビティーで駆動し, 実行結果を観察することによって, モデルのテスト・ベースでの検証をおこなうための仕組みについて考える.

状態機械はコード生成によく適合するので, 特に組み込み開発向けのモデル駆動開発のためのツールで手厚くサポートされている. 実際, 適切な状態機械図を描けば, それらのツールを利用して状態機械を実行することが可能である. ただし, 本報告での文脈においては, そもそもソフトウェアだけがモデリングの対象ではないこともあり, モデルは最終成果物に結びつけることを前提に作成されるのではなく, あくまで, 仕様の明確化や分析・設計の結果をできるだけ曖昧でない形式で文書化するという意味を持っているので, モデル駆動開発を主眼とするツールに見られるような, モデルを陽にプログラム言語のソースコードに変換し, コンパイルして実行するという方式はあまり適当でないと思われる. もちろん, 実際の開発プロセスでは, ソフトウェアで実現される部品に対して, その後のコード生成に結び付けていってもよいのであるが, これはSysMLでのモデリングとは異なるレベルのものであると考えるべきである.

モデリングをおこなう環境で, 作業者のモードを切り替えることなしにモデルの実行がおこなえるツールの例としてMEX[3]があげられる. MEXは本来, 開発中のモデルをインタラクティブに動作させて, 振る舞いの確認をおこなうものである(モデルをデバッグする, と称されている)が, アクティビティー図の実行が可能であり, それを状態機械図と連携させての動作も可能である. 実際にMEXを使用してみたところ, 実行エンジンの機能としては, 本報告での目的にあった利用が可能であることが確認できたため, 本報告で議論してきたような検証のためには, モデルを直接解釈し実行する(もしくは, そのように振舞う)処理系をベースに検証のためのシステムを構築するのが適当であると考える.

ただし, テストケースを逐一アクティビティーとしてモデリングするのは, テストケースが互いに似たものであることを考慮すると, 繰り返しが多く, 効率が良くない作業になるであろう. それを防ぐためには, モデル・ベース・テストの手法を用いて, テストスイート自体をモデル化し, テストケースを生成する等の工夫が必要であると思われる.

6. おわりに

モデリング作業での誤りや, 仕様自体の問題を発見することを目的として, SysMLで記述されたモデルについて, それ自身を, 最終成果物とは独立に検証することの利点やその実現についての議論おこなった. 次のステップとしては, これらの考察に基づいて, 実際にモデルの検証をおこなうためのツールを作成し, 実際の開発プロジェクトにおいて評価をおこなうことを予定している.

参考文献

[1] "OMG Systems Modeling Language V1.0", Object Management Group, Sept., 2007.

[2] OMG SysML, http://www.omgsysml.org/

[3] MEX, http://www.haifa.ibm.com/projects/software/ple/mex/

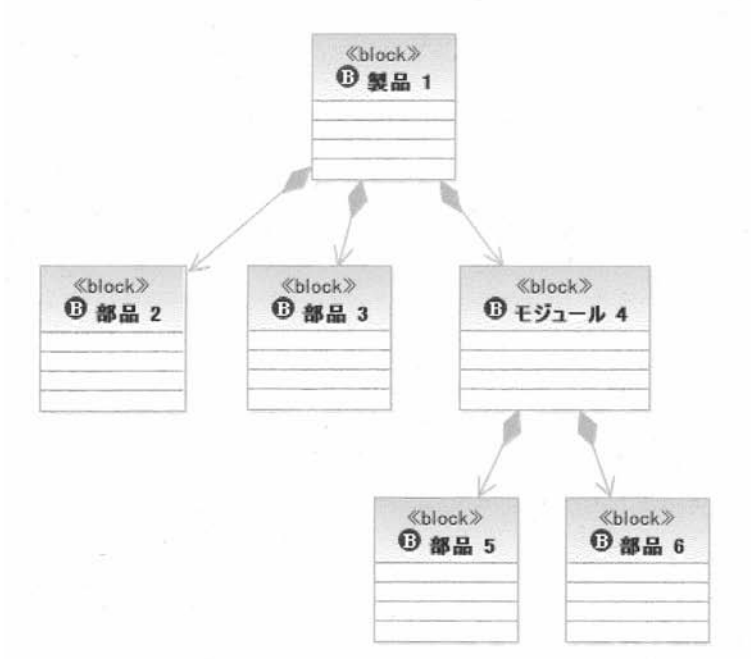

図 1. ブロック定義図

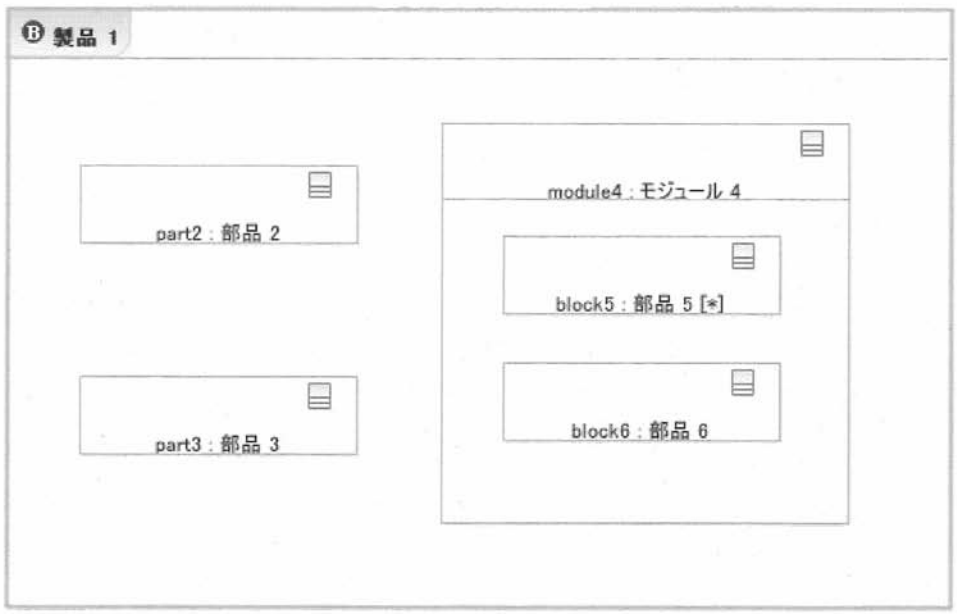

図 2. 内部ブロック図

図 3. 制約ブロック

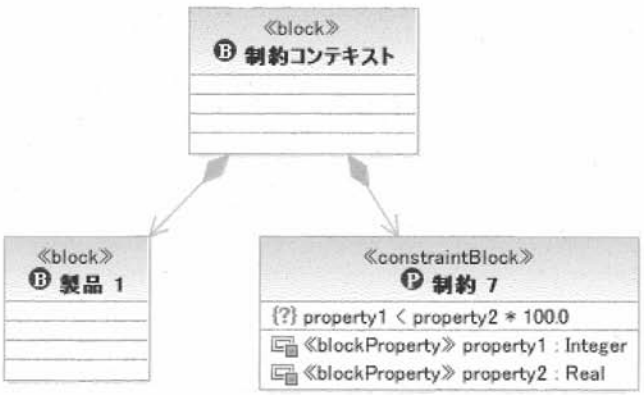

図 4. 制約のコンテキスト

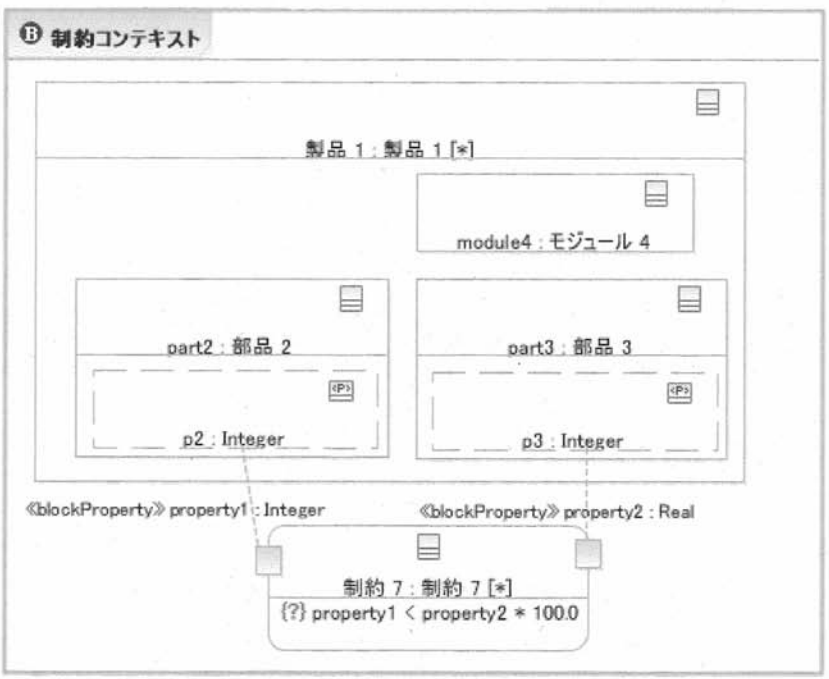

図 5. パラメトリック図

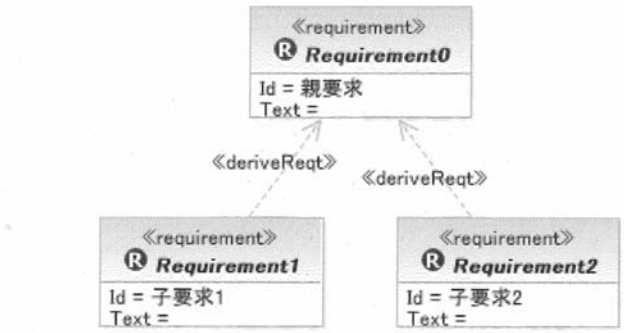

図 6. 要求図

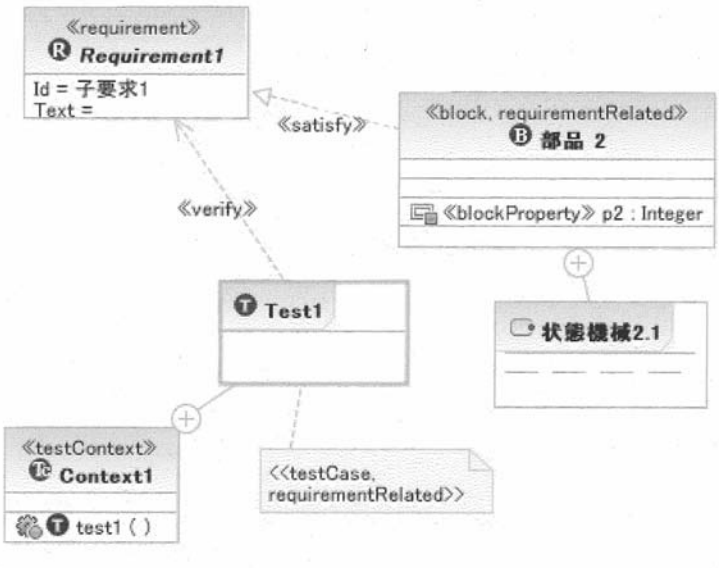

図 7. テストを含む要求図