B-021

モデル変換による組込みシステム開発事例
Embedded System Development using Model Transformation

伊藤 邦彦† 松浦 佐江子‡
Kunihiko Ito Saeko Matsuura

1. はじめに

組込みシステム開発では，センサーやアクチュエータといった機械部品を制御するソフトウェアを開発する．システムへの要求に対して，それを実現するハードウェア構成は多様であり，様々な組み合わせをシステムの利用環境の条件に応じて選択しなければならない．しかし，個々の機械部品の構成や能力によりシステムの特性が変化するため，本来のシステムへの要求を満たしている範囲において，ソフトウェアによる制御を調整する必要があり，多様なハードウェアとソフトウェアの組み合わせを実機ですべて確認することはできない．近年注目されている MDA(Model Driven Architecture)は，ソフトウェアおよびハードウェアアーキテクチャに依存しないモデルを作成し，モデル変換により対象ドメインに特化させることによって，組み合わせの多様性の解決を目指している． 一方，iUMLite[1]のような実行可能モデルではクラス図，ステートマシン図，アクション記述言語によりモデルレベルでの実行が可能である．本研究では，システムへの要求をハードウェアや環境に非依存なレベルから UML(Unified Modeling Language)のアクティビティ図を用いて段階的に開発し，このモデルから実行可能モデルを自動的に定義して，シミュレーションすることにより，多様なハードウェアに対するソフトウェアの制御を実機で実験することなく開発する方法を提案する．本稿では迷路探索ロボットの開発実験を通して，本方法の有効性を議論する．

2. システム開発プロセス

業務系システム開発においては，要求を分析する際，利用者や外部システムを含めた業務フローを十分に分析することが重要である．組込みシステムにおいても，組込み機器を含めたシステム全体の作業フローを分析し，その中での組込み機器の要件を検討する必要がある．例えば，迷路探索ロボットの場合では，「自律走行可能なロボットが，複雑に入り組んだ道を辿って，ゴールまで辿り着くこと」がシステム全体の要求である．すなわち，「迷路探索の解法を自律走行可能なロボットが実現する」ことが要求されていると考えることができる．しかし，ここで問題となるのは，迷路探索の解法を実現可能なロボットの要件はセンサーの種類，数，位置等，多様であり，全ての場合を実機によって確認することはできない．また，ハードウェア要件決定の根拠が明確にならないと，システム全体に対する要求自体が変化した場合に対応が困難となる．本研究では，これらの問題を解決するために，システム全体の振舞いを UML のアクティビティ図とクラス図を用いてモデル化し，ハードウェア要件を特定しながら，実行可能モデルに変換可能なモデルにまで段階的に詳細化する．そして，このモデルから，実行可能モデル（ステートマシン図）を自動的に作成して，シミュレーションを行うことにより，これらの多様性をシステム全体に対する要求の範囲内で確認する．図 1が提案方法の開発プロセスである．

†芝浦工業大学大学院 工学研究科 電気電子情報工学専攻
Division of Electronic Engineering and Computer Science, Graduate School of Engineering, Shibaura Institute of Technology
‡芝浦工業大学 システム理工学部 電子情報システム学科
Department of Electronic and Information Systems, College of Systems Engineering and Science, Shibaura Institute of Technology

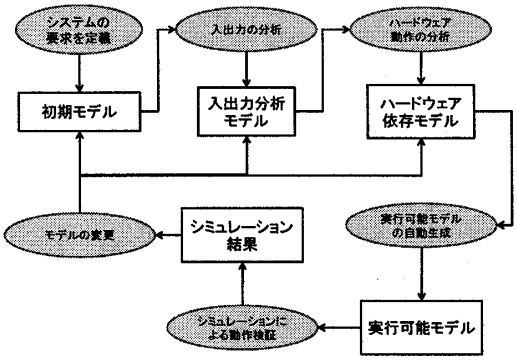

図 1 開発プロセス

3. 迷路探索ロボットの開発事例

3.1 概要

迷路探索の解法としては，右手法（または左手法）やその拡張法等がある．右手法とは，右の壁沿いに進むことで迷路を探索する方法である．この方法は平面的な迷路であれば，必ず出口にたどり着くが，開始位置または終了位置が迷路中央にある場合は，この方法ではゴールにたどりつけないという欠点がある．今回は開始位置，終了位置ともに迷路の角にあるものとし，右手法により探索が成功するものと仮定する．また，右手法は，現在位置から右側の壁の有無を確認するだけで次の行動を決定することができるので，制御のソフトウェアが単純に記述できることから，今回は迷路探索の解法として，右手法を採用する．

3.2 初期モデル

迷路のモデルは， 図 2に示すように，区画の形状が正方形の平面迷路のモデルとする．このモデルでは迷路は，方向・座標・開始位置・終了位置を属性として持つ．迷路探索の振舞いの初期モデルでは迷路の探索方法と終了位置での停止方法について決定する．終了位置の判定方法として開始位置から終了位置との座標の差を事前に与え，移動量を記録することで判断できるもとする．これらの方法を迷路探索する主体の固有の振舞いに依存しない形でアクティビティ図を用いて記述する．図 3のアクティビティ図は終

了位置でなければ右手法による探索を行う迷路探索を行う主体のアクションフローを定義している．右側の壁に沿って移動するために，現在の位置から認識する壁の方向と進む向きについての基本的なアクション（右方または前方を触る，右方または前方に進む，左に方向転換する）により右手法のアルゴリズムを定義する．

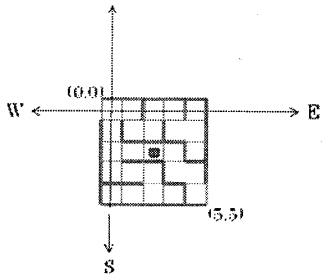

図 2　迷路の初期モデル

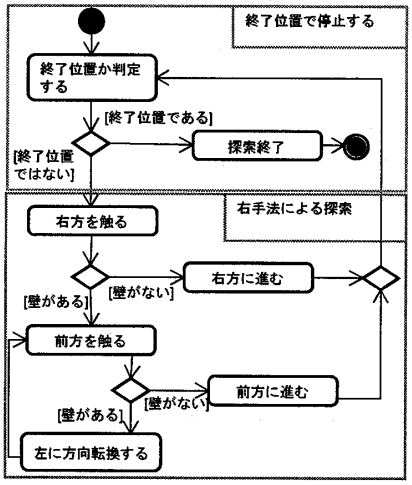

図 3　迷路探索の初期モデル

つぎに，迷路を探索する主体を自律走行可能なロボットとし，その振舞いによって，このアクティビティ図を詳細化する．図 2の迷路をロボットが探索するために，ロボットは，方向と座標をその属性としてもち，前進により進む単位（移動量）を迷路の一区画の幅として，つぎのように振舞いを詳細化する．まず，図 3で定義した基本的なアクションを探索パーティションに置き，これらのアクションの内，入力と出力に関わるアクションをロボットのアクションで置き換える．ここでは探索パーティションの色づけされたアクションがこれに相当する．どのように詳細化したかを明確にするために，アクティビティ図に入力および出力のパーティションを定義する．入出力に関する分析結果のアクティビティ図が図 4である．

「右方を触る」「前方を触る」といったアクションはロボットの「壁の有無を調べる」というアクションによって実現する．「進む」というアクションは，迷路の一区画の中心から，ロボットの前方向にまっすぐに，迷路の幅だけ「前に進む」というアクションによって実現する．「右（左）方に進む」というアクションは，迷路の一区画の中心から，位置を変えずに，ロボットの前面の向きを座標軸に沿って−90 度（90 度）回転する「右（左）に向く」というアクションと「前に進む」というアクションの連接として実現する．これらのアクションによるロボットの属性（方向・座標）の変化をオブジェクトノードにより定義する．

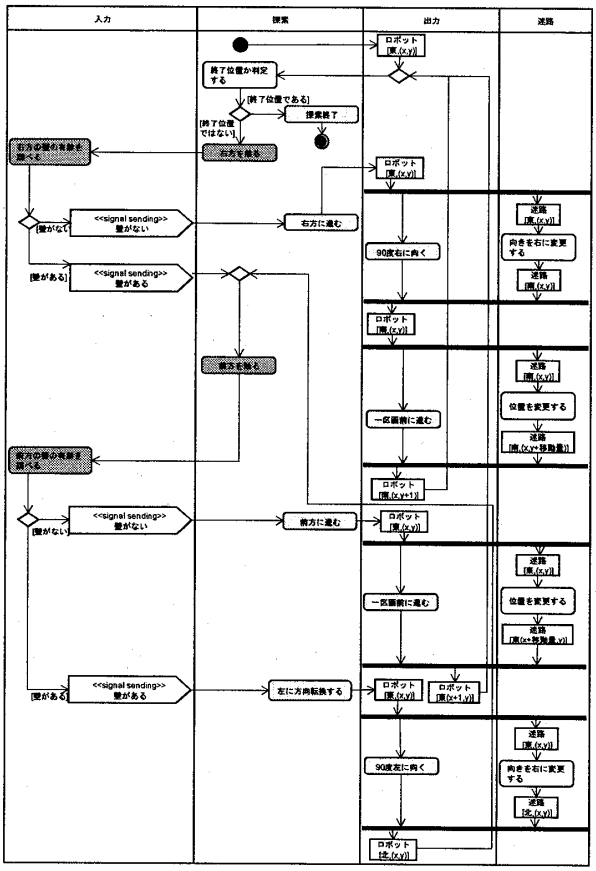

図 4　入出力分析モデル

ここで，組込み特有の問題が発生する．実機を使用して迷路探索を行う場合、実機では外部環境を知覚するセンサーの入力から移動を行うアクチュエータによって出力を行う．コンピュータ上でのアルゴリズムの実装では、この入力および出力を3.2節で述べた迷路のモデルにより決定すれば十分である。しかし、実機で行う場合は、迷路のモデルはハードウェアおよび実際の迷路に依存する。入出力は実機の作り出すものであるから、ロボットが認識している自己の属性値の変化とは異なる結果になることがある。この誤差を考慮して、全体の目的を達成できるようにソフトウェアがロボットを制御する必要がある。そこで，図 4に示したように，ロボットが出力を行う際に，実際の出力結果を迷路上に記録するアクションを「迷路」パーティションを作成し，そこに定義する．これはロボットが「右に向く」であれば「向きを右に変更する」というアクションと，迷路の属性を用いたオブジェクトノードにより定義する．

「入力」「探索」「出力」はロボットのソフトウェアの世界であるが，「入力」や「出力」は直接ロボットのハードウェア機器へ作用するものである．ここで生じる誤差をモデルに採り入れることでシミュレーションによる問題点の発見と，問題への対処方法を議論できるようにする．た

だし，今回は「入力」に関する誤差のモデルは作成していない．

3.3 ハードウェア要件

初期モデルを実現するためにはセンサーの能力として壁の有無を判断しなくてはならない．また，アクチュエータの能力として，前進，その場での左右回転を行えなくてはならない．今回の実機は LEGO MINDSTORMS NXT[2]を使用することとした．使用可能なセンサーには，タッチセンサー，超音波センサー，サウンドセンサー，光センサーの 4 種類があり，壁の有無を判断できるものとしては，タッチセンサーと超音波センサーが利用できる．

今回は超音波センサーを採用した例を示す．

3.4 外部環境とハードウェアの制約

外部環境とハードウェアには以下の制約が存在する．

- 区画幅 >= 車体幅
- 壁の高さ >= センサーの高さ
- 区画幅 == 移動量

これらの制約を満たしているときに動作可能となる．実装時にはこれらの制約を満たす環境・ハードウェアを製作する．

3.5 クラス図の定義

外部環境である迷路と迷路探索ロボットのクラスの定義を行う．ここではシステム動作後に更新される属性とシステム動作時に更新されない属性を分けて考える．図 5の迷路探索ロボットクラスは区画座標 X，区画座標 Y，方向をシステムの動作時に更新される属性である．迷路クラスでは，ロボットの絶対座標として、ロボット座標 X，ロボット座標 Y を持ち，ロボットの方向を角度で持つ．ロボットの区画座標 X・区画座標 Y は絶対座標に対する区画番号である．また迷路探索ロボットクラスの移動量，幅，高さ，長さ，センサーの高さ，迷路クラスの区画幅縦，横，壁の高さ，迷路の構造はシステムの動作前に設定される属性である．

迷路探索ロボットの方向は誤差なく左右回転を行うことを前提としているため東西南北の 4 方向を扱うものとして列挙する．それに対して，迷路クラスではロボットの座標を絶対座標で扱う属性とそれに対応した区画座標を持ち方向は角度を扱う．

3.6 ハードウェア・外部環境依存モデル

図 4のアクティビティ図からハードウェア固有の動作のアクティビティ図を作成する．図 6は超音波センサーを使用するもとのとして入出力を分析したものである．超音波センサーは壁との距離を取得することにより非接触で壁の有無を判断することができるが，センサーを前方に取り付けるため，壁の有無を判断したい方向を向かなくてはいけない．これは実装予定のハードウェアは側面に取り付けると壁の高さとセンサーの高さの制約に違反してしまうためである．センサーが前方にあるため「右方を触る」アクションでは「90 度右に向く」アクションを行った後に「壁との距離を取得する」アクションを行わなくてはならない．

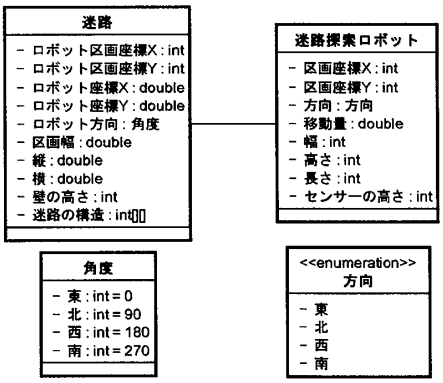

図 5　クラス図

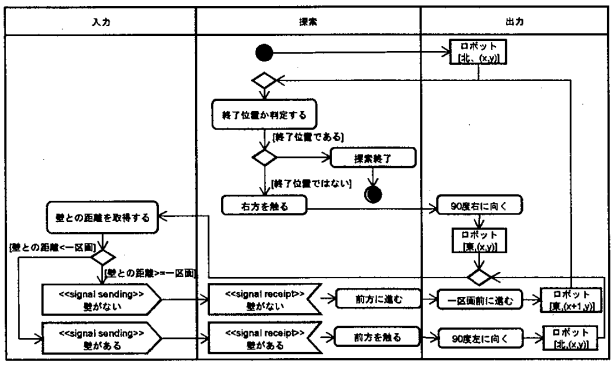

図 6　超音波センサー依存モデル

3.7 実行可能モデル

今回は，実行可能モデルを記述，実行するツールとして iUMLite[1]を使用する．実行可能モデルではクラス図・ステートマシン図また各状態にアクション言語により具体的な操作を記述する必要がある．図 6のアクティビティ図から実行可能モデルに必要なクラス図，ステートマシン図，アクション言語による記述を生成する．

3.7.1 クラス図

アクティビティ図の「入力」「探索」「出力」の各パーティションは迷路探索ロボットのソフトウェアの構成を表現している。すなわち、クラスに相当すると考え、各パーティションをクラスに割り当てる。ステートマシン図は識別可能なクラスの状態の遷移を表すことから、各パーティション内のアクションに注目し、状態を識別する．今回は入力パーティションを Sight，探索パーティションを Search，出力パーティションを Body クラスと外部環境クラスを Maze クラスとした（図 7）．Maze クラスでは図 4の迷路パーティション内にあるアクションをメッソドとして持つ．forward メソッドでは「位置を変更する」アクションを行う．このメソッドでは Maze の属性を移動量と方向に応じて値を変更する．また，Executable UML ではクラス名とクラス番号，クラスの略称を書く．番号，略称を使うことでアクション内において簡潔に表現することができる[3]．今回略称はクラス名を大文字にするものとし，クラス番号は iUML が自動で割り振る値を使用する．関連名は

関連を曖昧なく識別するために「R5」のような関連番号が使用される．

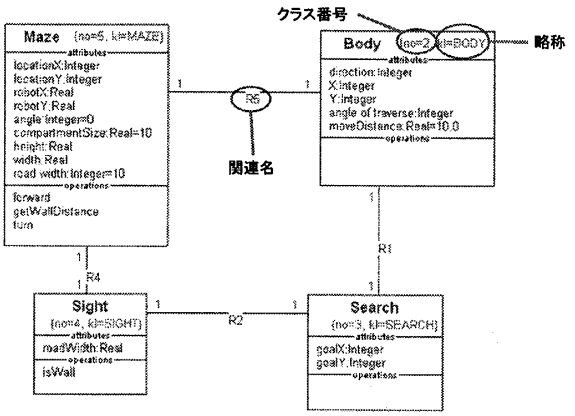

図 7　iUML クラス図

3.7.2 ステートマシン図

図 6の各パーティションに対応するクラスにそれぞれステートマシン図を作成する．Executable UML でのステートマシンの構成要素は状態・イベント・遷移・プロシージャである．状態はインスタンスのライフサイクルでの段階を表す．イベントは出来事を表す．遷移は，所定の状態において特定のイベントを受けとる時に，どの新しい状態に到達するかを定義する．プロシージャはある状態に到達した時に遂行すべきアクティビティまたは操作である．プロシージャはアクションから構成される．

ハードウェア依存モデルのアクティビティ図から実行可能モデルのステートマシン図への変換ルールを表 1に示す．以降探索パーティションを例に図 8のステートマシン図への変換を説明する．表 1のルール 1 は探索パーティション内のアクションを状態とする．名前の対応は表 2となっている．ルール 2 は遷移を生成するルールである．右方を触るアクションからエッジをたどると次に同じパーティションに到達するアクションは前方に進むアクションと前方を触るアクションとなり，RightTouch 状態から Forward 状態と ForwardTouch 状態への遷移を生成する．ルール 3 はプロシージャ内のシグナルアクションに変換する．迷路探索ロボットでは入力クラスの壁との距離を取得する状態のプロシージャのアクションとなり，シグナル受信がある探索クラスにシグナルを送信する．ルール 4 はイベントを決定する．アクティビティ図では右方を触るアクションから前方に進むアクションの間に「壁がない」シグナル受信が存在することから，RightTouch 状態から Forword 状態への遷移のイベントが「nonWall」となる．ルール 5 はオブジェクトノードの状態変化から状態のプロシージャを決定する．出力パーティションの 90 度右を向くアクション後にロボットの状態を[北,(x,y)]から[東,(x,y)]に変更している．このことから出力クラスの 90 度右を向く状態のプロシージャにて属性である方向を変更するアクションを記述する．ルール 5 はアクションの条件に関する記述へ変換する．探索クラスパーティションの終了位置か判定するアクション後のガードは図 8の CheckingGoal 状態の if 文に対応する．この if 文では現在の位置が終了位置であるかどうかを判定し，終了位置であれば「goal」のシグナルを送信する．終了位置でなければ「notGoal」のシグナルを送信する．

表 1　モデル変換表

	アクティビティ図	ステートマシン図
1	対象クラスのパーティション内のアクション	状態
2	対象パーティション内から次の対象パーティションへのエッジ	遷移
3	シグナル送信	シグナル送信直前のアクションに対応する状態のプロシージャ
4	シグナル受信	シグナル受信後のアクションに対応する状態に到達する遷移のイベ
5	オブジェクトノード	オブジェクトノード直前のアクションに対応する状態のプロシージャ
6	ガード	ガード直前のアクションに対応する状態のプロシージャ ガード後にシグナル送信がない場合は直後のアクションに対応する状態に到達する遷移のイベント

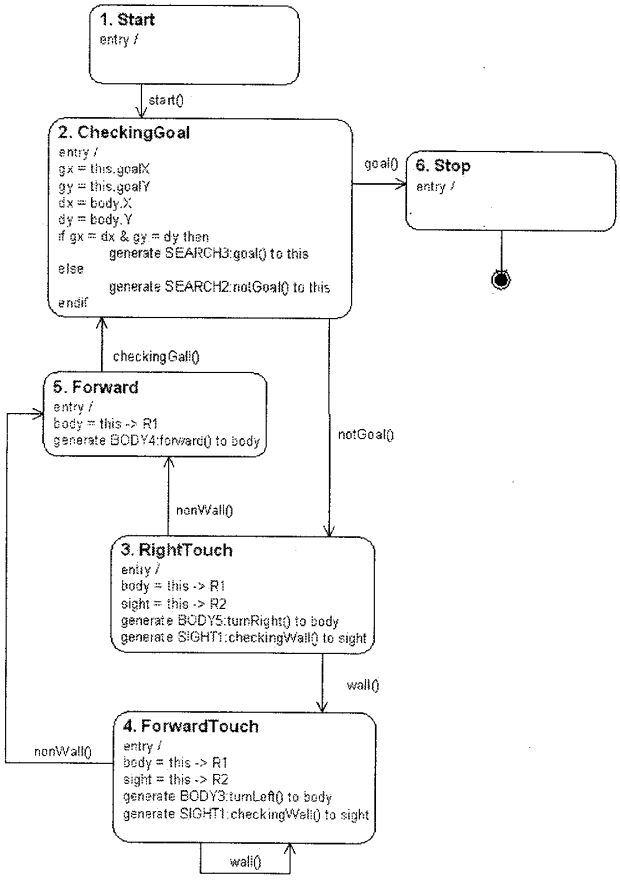

図 8　探索クラスのステートマシン図

表 2　アクションと状態の対応

アクション	状態
終了位置か判定する	CheckingGaol
探索終了	Stop
右方を触る	RightTouch
前方に進む	Forward
前方を触る	ForwardTouch

3.8 シミュレーションによる制御プログラムの検証

iUML のシミュレーションでは3.7節で述べたモデル以外に初期化メソッドとテストメソッドを定義する必要がある．初期化メソッドではインスタンスの生成および属性の初期化を行い，インスタンス間の関連を定義することをアクション言語によって記述する．テストメソッドでは起動するインスタンスを選択し，シグナル送信またはメソッド呼び出しを行うことでテストを実行する．また，属性値を任意の値にすることができるため，特定の状況を設定し，テストを行うことも可能である．

シミュレーション実行時にインスタンスの属性値およびステートマシンの状態を確認することができる．また，実行後に実行履歴として確認することができるため，属性値の変化を見ることによって検証を行うことができる．

3.8.1 理論値による実行

外部環境である迷路を誤差なく一区画進み，90 度の左右回転することで右手法のアルゴリズムで迷路探索が可能であるかを検証できる．ロボットが認識している区画座標と外部環境における区画座標が異なることなく終了位置まで探索することができるかを検証する．ここでは探索中の履歴を見ることにより右手法を正しく実行し，終了位置で探索を終了することを確かめる．

3.8.2 出力の誤差の影響

実機では必ずしも理論値通りの出力を行うわけではない．理論値と異なった移動量を初期設定時に入力することにより前進動作時に起こる移動距離の誤差について検証を行った．シミュレーション方法として，壁を超えて移動した時シミュレーションを終了することとした．その結果ロボットが終了位置に辿りつく前にシミュレーションが終了した．たどり着くことができない原因としてロボットの認識位置と実際の位置に差異が生じたことが挙げられる．また，超音波センサーは壁の有無を一区画以内に壁があるかないかで判断しているため，区画幅を超えて移動することにより壁と衝突する状態になる現象が起こった．

4. 考察

4.1 今後の課題

初期モデル，入出力分析モデル，ハードウェア依存モデルの 3 つのモデルを段階的に作成し，モデル変換を行うことにより実行可能モデルを作成した．実行可能モデルによって動作の検証を行うことができた．検証の結果からモデルを修正する場合は，実行可能モデルを修正することなくアクティビティ図を変更することによってシステムの振舞いを修正することができる．段階的にモデルを作成したことにより，修正の内容によってどのモデルを修正するべきであるかを特定することができる．ハードウェア固有の問題，またはハードウェアの変更であればハードウェア依存モデルを修正する．入出力アクションの問題である時は，入出力モデルの修正を行う．3.8.2 節で述べた誤差の影響を修正する方法として入力回数を増加することにより，誤差を修正することができる．この方法を採り入れるためには入出力分析モデルを修正する必要がある．また，探索方法および終了方法を変更するには初期モデルの修正が必要であることがわかる．

しかし，一つのモデルを修正すると，他のモデルに影響を与えるためその他のモデルも修正を行わなくてはならなくなる．現在段階的にモデルを作成するときにハードウェア・外部環境の要件を入れ込んだモデルとなっているため，解決は困難である．これを解決するためにはハードウェア・外部環境を分けて定義することが必要であると考える．今後段階的に作成したモデルの整合性を保つことが必要となる．

4.2 関連研究との比較

組込みシステムの特徴は，リアルタイム性や実世界の物理事象を扱うことである．システムが実世界に影響を及ぼし，かつシステムは変化する実世界に適切に対応しなくてはならない．そのため，実世界に対するリアクティブ性を考慮した，システム外部にある実世界の影響を分析する手法が提案されている[4]．本稿で扱う迷路探索ロボットの場合も，実世界におけるシステムの振舞いそのものをモデルに採り入れ，分析する必要がある．

組込みシステムの個々の製品プラットフォームに依存しない論理的なモデル記述を行い，それをシミュレーション実行することで製品シリーズ全体の開発効率の向上を図る提案[5]や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ルに対してテストを実行することにより検証を行う方法が提案されている[6]．しかし，ハードウェアの要件を事前に決定しているため，ハードウェアの変更については議論されていない.本稿では多様なハードウェアから形式的に要件を採り入れていく方法を提案した．

5. おわりに

迷路探索ロボットの開発事例を基にシステム全体の振舞いをアクティビティ図とクラス図により定義し，段階的に詳細化を行うことによってハードウェア・外部環境固有の問題を分離して分析を行った．また実行可能モデルを生成することによってシミュレーションを行いシステム全体の振舞いの検証と修正を行う方法について述べた．

参考文献

[1]Kennedy Carter, “iUMLite” http://www.kc.com/
[2]LEGO MINDSTORMS , http://www.legoeducation.jp/mindstorms/
[3]スティーブ J.メラー，マーク J. バルサー”Executable UML”翔泳社(2003)
[4]鷲見 毅, 平山 雅之, 鵜林 尚靖, 組込みシステムにおける外部環境の分析, 情報処理学会研究報告 ソフトウェア工学研究会報告, Vol.2004, No.118 (2004).
[5]伊藤 恵, 久保秋 真, 組み込みソフトウェア開発のための仕様シミュレーション, 情報処理学会研究報告, Vol.2002, No.64 (2002).
[6]広瀬　紳一,SysML モデルの検証についての考察,情報処理学会研究報告　ソフトウェア工学研究会報告 Vol.2007 No.107(2007)