B-019

UML-PROMELA 変換器を用いた ZigBeeIP/RPL プロトコルにおけるノード探索仕様の検証

Verification of the Node Search Specifications in ZigBeeIP/RPL Protocol using UML-PROMELA Converter

後藤 亮馬 †　和崎 克己 ††
Ryoma Goto　Katsumi Wasaki

1　はじめに

ソフトウェアの仕様を検証する手法としてモデル検査がある．モデル検査によるシステム検証では，検証対象を形式的に記述したモデルとして記述することが必要となる．上流工程において，モデル記述からモデル検査器用のプロセスに変換する手法は広く研究されており，UML-PROMELA 変換器 [1][2] について著者らは種々に提案を行ってきた．

本研究では UML-PROMELA 変換器を用いて，RPL(IPv6 Routing Protocol for Low-Power and Lossy Networks)[3] のモデル化，検証を行った．上位設計向けの準形式化されたこれらの図を使用してモデルを記述することで，検証したいノード数や通信状況の変化に応じて自動コード生成が行われるため，スケーラブルな対応を迅速に行うことができる．今回作成した UML モデル図と，それから自動生成された PROMELA コードを用いて RFC6550 のノード探査部分を中心にデットロック検証を行い，エラーが無いことを確認した．

2　モデル検査

2.1　概要

モデル検査はソフトウェアなどの仕様が正しく動くことを検証する手法の一つである．検査対象となる仕様の振る舞いをモデルという専用プロセスとして記述し，検査器を用いて初期状態からとりうる状態を網羅的に検査する．仕様に要求される性質は各ステップにおける表明，あるいは時相論理式などによって記述することができ，性質に反している状態が発見された場合はエラーとして報告される．この手法は設計段階から適用することができ，開発コスト削減に繋がることが期待されている．複数のモデル検査器が存在しているが，本研究ではモデル検査器 SPIN[4] を使用する．

2.2　モデル検査器 SPIN

モデル検査器 SPIN(Simple Promela INterpreter) はベル研究所の G.J.Holzmann 氏によって提案されたモデル検査ツールである．この検査器は通信プロトコルの検証を目的として提案されたが，通信プロトコル以外の検証にも使用することができる．検査する振る舞いであるモデルは，PROMELA(PROtocol/PROcess MEta LAnguage) という専用の仕様記述言語にて記述する．要求する性質の記述には，モデルに直接 assert 文を書き込む方法の他に LTL 式 (Linear Temporal Logic) を用いることができる (図 1)．

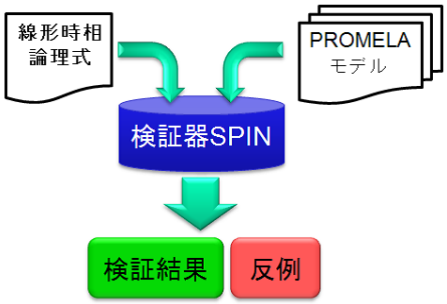


図 1　モデル検査器 SPIN

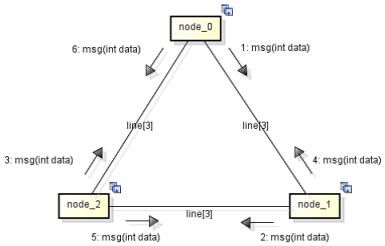


図 2　コミュニケーション図の記述

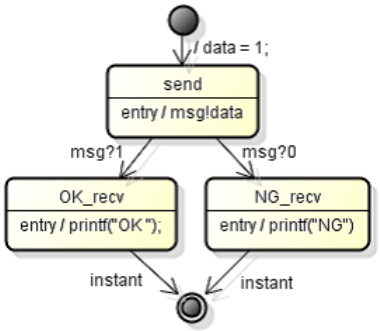


図 3　ステートマシン図の記述

3　UML-PROMELA 変換器

UML-PROMELA 変換器は坂本ら [1] によって提案された変換器である．UML 図 [5] のコミュニケーション図とステートマシン図を用いてモデルを記述し，PROMELA コードに自動変換する．図の記述には astah[6] を用い，XML ファイル経由で変換を行う．

3.1　UML 図の記述方法

コミュニケーション図では通信状態を記述する．各ノードの相互接続状況と，どのようなメッセージを流す

† 信州大学大学院理工学系研究科，Graduate School of Science and Technology，Shinshu University.
†† 信州大学工学部，Faculty of Engineering，Shinshu University.

のかを記述する．図 2 の記述では，node_0 と node_1 と node_2 が line というバッファ容量 3 の通信経路で接続されていて，msg という通信名で int 型の変数 data を送信し合うという通信状況を表している．

ステートマシン図では通信機器がどのように振舞うのかを記述する．メッセージを受信した際の振る舞い，タイムアウトが発生した際の振る舞いなどを記述する．図 3 では，開始状態から変数 data に値 1 を代入して send 状態に遷移した後，通信名 msg で packet を送信して受信待機し，msg にて値 1 を受け取った場合は OK_recv 状態に遷移し OK を出力，あるいは値 0 を受け取った場合は NG_recv 状態に遷移し NG を出力する振る舞いを表した記述である．instant の記述は強制遷移を表し，メッセージ受信などに関わらず状態を遷移させる場合に使用する．

3.2 PROMELA コードへの変換

図で記述したされたモデルは以下のように PROMELA コードに自動変換される．図 2 の通信状況は各ノードをつなぐチャネル，変数を用いて定義される．

```
chan line_1_msg = [3] of { int };
chan line_2_msg = [3] of { int };
chan line_3_msg = [3] of { int };

active proctype node_0{};
active proctype node_1{};
active proctype node_2{};
```

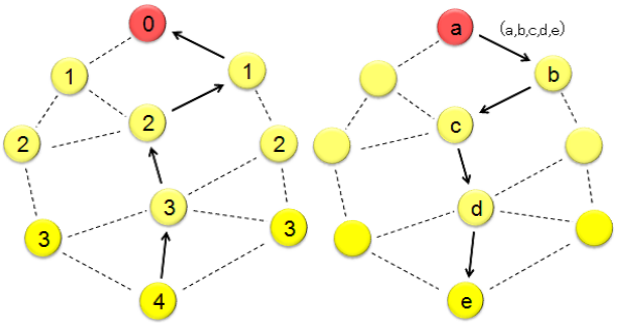
チャネル名は通信経路名_識別番号_通信名という規則で命名される．バッファ容量 3 の int 型は [3] of { int }; が表しており，active proctype〜の部分がノードを表している．data という変数名はノードの振る舞い記述として出力される．

```
mtype node_0_state = send;

active proctype node_0{
    int data = 1;
    int arg0_1_send;

    line_1_msg!data;
    do
    ::line_2_msg?arg0_1_send ->
        if
        ::arg0_1_send == 1 ->
          node_0_state = OK_recv;
          printf("OK");
          break;
        ::arg0_1_send == 0 ->
          node_0_state = NG_recv;
          printf("NG");
          break;
        fi;
    od;
};
```

図 3 の振る舞いは使用される変数宣言，メッセージ送信，do 文での受信と出力で表現される．また，各ノードの状態を示す変数としてグローバル変数が定義される．全コードは長いので省略するが，大まかな流れとして上記のようなコードが出力される．line_1_msg!data; で送信を行った後，繰り返し文 do 文にて受信を待機する．受信が発生した場合は if 文にて受信内容が 1 か 0 かを判定する．arg0_1_send はこのときに使用される判定用の自動生成変数である．中身が 1 だった場合は現在の状態 node_0_state を OK_recv に変更し printf 文にて OK を出力し break 文で do 文を抜けて終了，中身が 0 だった場合は現在の状態 node_0_state を NG_recv に変更し printf 文にて NG を出力し break 文で do 文を抜けて終了する．



(a) 上り方向の通信　　(b) 下り方向の通信

図 4　DAG での通信 (数値は rank 値)

4 ZigBeeIP/RPL Protocol

UML-PROMELA 変換器を用いて作成されたリングアルゴリズムなどの従来のモデルは正しい結果が得られている．より実仕様に近いモデルを用いた検証を行うという点から ZigBeeIP にて使用されるルーティングプロトコル RPL[3] を今回の検証対象とした．

ZigBeeIP は ZigBee ネットワークを IP ネットワークに接続するための規格である．UDP/TCP で通信を行い，IPv6 のデータパケットを交換するための通信プロトコルである．ZigBee は近距離無線規格の一つであり，主にセンサネットワークを目的としている．転送距離は短く，通信速度も遅いが安価，省電力であるという特徴を持つ．

RPL(IPv6 Routing Protocol for Low-Power and Lossy Networks)[3] は ZigBeeIP にて使用されるルーティングプロトコルである．各通信機器は DAG(Directed Acyclic Graph) と呼ばれる非環式のツリートポロジーを形成することで通信を行う．

DAG に参加したい通信機器は，まず DIS(DODAG Information Solicitation) を送信し DAG への参加表明を行う．DIS を受けた通信機器は DIO(DODAG Information Object) を返信し，根のアドレスや自分の rank 値を伝える．rank 値は根からその通信機器までの仮想的な距離を表す値で，この値は小さいほど根への通信コストが低いものとされる．DIO の中から rank 値が最も小さいものを親として選択し，DAO(Destination Advertisement Object) を根に送信する．DAO には自分が親としてどの機器に接続しているかなどの情報が含まれており，根はこの情報を元にソースルーティングを行う．最後に根から DAG に接続したことを伝える DAO-ack を受け取り，接続が完了する [7]．

上り通信 (根へ向かう通信: 図 4(a)) を行うときは，親を辿ることで情報を伝達する．下り通信 (葉へ向かう通信: 図 4(b)) は DAG 形成時に各機器から収集した情報を元に根が伝達ルートを選択し，各機器はこのソースルーティングに従って情報を伝達する．

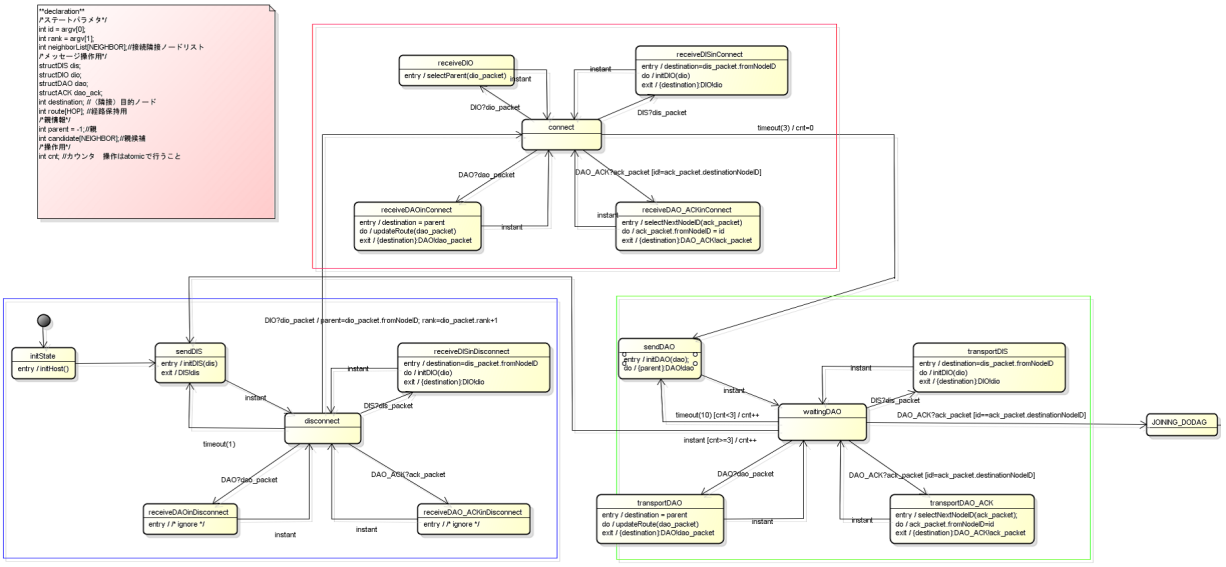


図 5　ステートマシン図: ホストノード全体の振る舞いモデルの記述

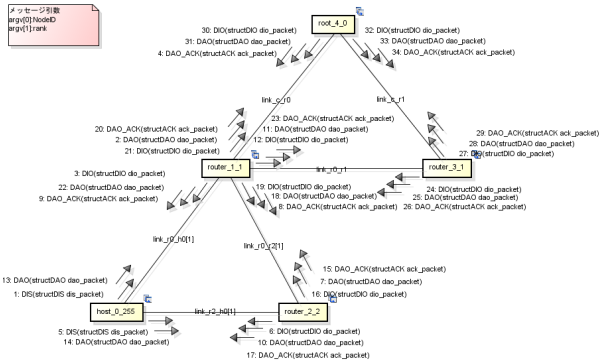


図 6　コミュニケーション図: 通信接続状況

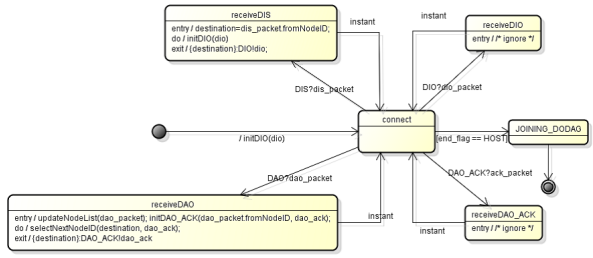


図 7　ステートマシン図: コーディネータのモデル記述

5　UML 図を用いたモデル化

UML-PROMELA 変換器を用いて対象のモデル化を行う．どの DAG にも接続していない末端の通信機器ホストが，親を決定して根に接続を報告し，DAG に接続するまでのシナリオをモデル化する．変換器の特性上，通信機器の個数の動的な変化が記述できないため，通信機器の個数を限定した状態で検証を行う．また，rank 値は既に与えられているものとする．各図は坂本らの UML 図に対する追加記述の規定 [1] に準拠した形式で作成する．

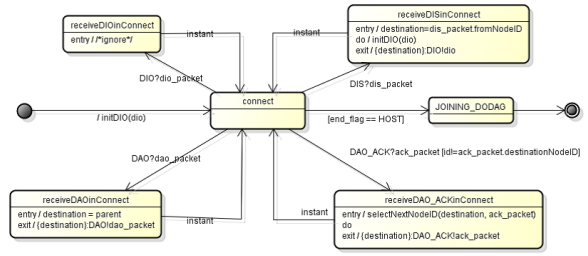


図 8　ステートマシン図: ルータのモデル記述

5.1　コミュニケーション図における記述

ノード間での通信接続状況を記述する (図 6)．DAG の根が 1 つ，節が 3 つ，葉が 1 つの状況を想定しモデル化を行う．各ノードは link〜という名前のバッファ数 1 の通信経路を持ち，その通信経路内に DIS，DIO，DAO，DAO_ACK の 4 種類のメッセージが流れることを記述する．ノード名に続けてノード番号を記述し，さらにその後ろに rank 値を記述する．

5.2　ステートマシン図における記述

DAG の葉であるホストは図 5 のように記述する．状態は大きく分けて 3 つあり，DAG に接続していない disconnect，親を決定したが根に状態が伝わっていない connect，根に状態が伝わった確認応答を待つ waiting-DAO_ACK 状態である．DAG への参加表明である DIS を送信した葉は disconnect 状態で隣接ノードからの応答メッセージ DIO を待つ．DIO を受信後は connect 状態に遷移すると共に親を決定する．他に親候補となるノードがいないか timeout カウント分待機した後，DAO を送信し waitingDAO_ACK 状態に遷移し受信確認応答を待つ．確認応答を待ちつつ timeout カウント分時間が経過した場合は DAO を再送する．DAO_ACK を受け取った場合は，メッセージの中身が自分宛であることを確認し，問題なければ終了状態に移行する．自分宛でなかった場合はもう一度 DIS を送信して disconnect 状態に遷移し，最初から接続をやり直す．

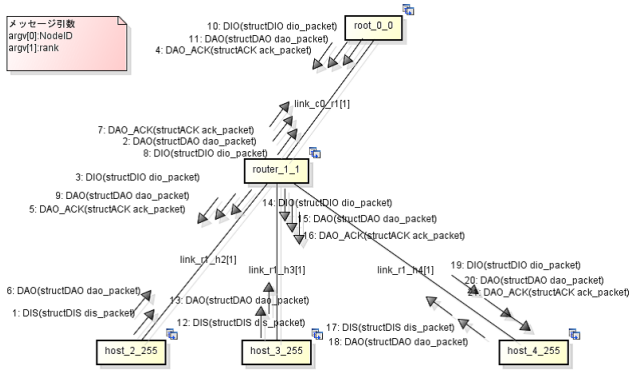


図 9　変更後の通信状況 (ホスト数 4)

DAG の根であるコーディネータの振る舞いは図 7 のように記述した．状態 connect でメッセージ受信を待機し，メッセージ受信後は対応した状態に遷移し，対応した振る舞いを行う．葉の機器全てが DAG に参加したことを検知した場合は，JOINING_DODAG 状態に遷移し，振る舞いを終了する．本来の振る舞いとしてはメッセージを受け取るたびに各メッセージに対応した行動をするが，今回想定する状況では DIS，DIO，DAO_ACK を受信することは無いため，状態数削減の観点から記述を省略している．

DAG の節であるルータの記述は図 8 である．コーディネータと同じように状態 connect でメッセージを待機し，受信に対応した状態に遷移し，対応した振る舞いを行う．受信状態で振る舞いを行った後は直ぐ connect に戻り，メッセージ受信を待機する．また，今回の状況では DIO は受信しないため，記述を省略している．

6　コード自動生成と検証結果

作成した図を用いてコードを自動生成し，各ノードが不正な状態で停止することがあるかどうかを調べるデットロック検査を行った (ケース [A])．また，各機器の振る舞いは変更せず，通信状況だけを変更した図 (図 9) を用いてホストを増やしていくとどのような傾向があるのかも検証した (ケース [B])．

検証環境は CPU:IntelCorei7L640，RAM:8.00GB，OS:Windows7proX64 である．

6.1　生成されたコード

自動変換によって出力された PROMELA コード行数は，ケース [A] で 1,091 行，ケース [B] で 671〜2,001 行であった．また，作成された変数はケース [A] で 140 個であった．図を使用せずに整備するには規模の大きいコードである．UML 図からコードへの変換にかかった時間は全て 5 秒以内であった．

6.2　検証結果

ケース [A] では，状態数 15,112，深さ 162，エラー無しという結果となり，検証にかかった時間は 0.842 秒であった．この結果は，このモデルにおいて各通信機器は処理を終えずに停止することが無いということを意味している．また，振る舞いが終了する状態が DAG を形成しホストが接続を終えた後のみなので，ホストの接続までのシナリオは必ず発生することが分かった．

ホスト数	深さ	状態数	検証時間 (s)
1	118	268	0.004
2	211	18,020	0.264
3	288	1,004,349	19.9
4	428	31,474,096	805
5	473	57,495,424	2,110
6	576	62,890,526	2,710

表 1　ホストを増やしていった結果

ケース [B] の検証結果を表 1 に示す．ホストの数を増やしていくと状態数，深さ共に増えていっているのが分かる．これは，節と根を繋ぐ通信経路が一つしかなく，一つの通信路に通信が集中する状況が発生しているので，受信待機状態が増えた結果，タイムアウト処理が増え，状態数が増えたものと考えられる．

7　まとめと今後の課題

UML-PROMELA 変換器を用いて RPL プロトコルのデットロック検証を行った．検証の際には機器の振る舞いであるステートマシン図はそのままに，コミュニケーション図だけを変更することで通信接続状況を変化させた．検証の結果，デットロックは検出されず，不正な状態で終了することが無いことが分かった．また，使用する通信経路が一つに集中することで，深さ，状態数が増えていく様子も観察できた．

UML-PROMELA 変換器は UML 図の記述から検証までを一貫したプロセスで検証を行うことができるツールである．モデル作成後に変更する必要があった場合でも，図を変更するだけで対応ができるため，PROMELA コードを直接書くことに比べてモデル作成のコストは抑えられた．しかし，通信が途中で途切れる状況や，通信機器の数が変化するといった通信状況が動的に変化した場合の検証にはまだ対応できていない．また，asssert 文や LTL 式といったモデルに要求する性質の記述は，自動変換後に使用者が直接コードに書き込む必要がある．通信接続状況の変化が発生する検証への対応，要求する性質の記述を含めた自動変換が今後の課題だと考えられる．

参考文献

[1] 坂本統, 和崎克己：“タイムアウト機構を有するメッセージ交換プロトコルの UML モデルと SPIN モデル検査”；FIT2013（第 12 回情報科学技術フォーラム）講演論文集, (B-021), 267-270, 2013.
[2] 宮本直樹, 和崎克己：“上流設計からモデル検査プロセスまでの一貫設計検証環境”；信学技報 (SWIM2011-19), 111(308), 7-12, 2011.
[3] RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, IETF RFC6550, 2012.
[4] Gerard J. Holzmann：“The SPIN Model Checker”；Addison-Wesley, 2004.
[5] UML Resource Page, The Unified Modeling Language(UML),the Object Management Group(OMG), http://www.uml.org/
[6] 株式会社チェンジビジョン, http://www.changevision.com/
[7] OKI テクニカルレビュー第 221 号 Vol.80 No.1, 70-73, 2013.
[8] 吉岡信和, 青木利晃, 田原康之：“SPIN による設計モデル検証”；近代科学社, 2009.
[9] 中島震：“SPIN モデル検査”；近代科学社, 2008.