

デザインパターンによる DCS シミュレーションと実装の統合化

— DCS ネットワーク構成情報の効率的実装法 —

旭川工業高等専門学校
北海道大学
モトローラ株式会社

○戸村 豊明
金井 理, 岸浪 建史
伊深 和浩, 上広 清, 山元 進

要 旨

分散制御システムにおける各制御ノードの状態遷移仕様を実装するために、本報では、これまで我々が提案してきた Statechart パターンのオブジェクト記述から、制御ノードの状態遷移仕様の形式記述へ変換した後、その形式記述をオープンネットワーク LonWorks における制御アプリケーション開発用言語である Neuron C で記述された実行可能コードへ変換するための手法を提案する。さらに本報では、この手法を実行するソフトウェアとして、Statechart コンパイラを開発する。

1. はじめに

近年、FA（ファクトリオートメーション）、BA（ビルオートメーション）、自動車内機器間高速通信の分野において、多数の制御ノードを相互接続したオープンネットワークを持つ分散制御システム（以下 DCS）が導入され始めている。

DCS に要求される各機能は 1 つのユースケース¹⁾として記述できる。ユースケースを構成する各機能ブロックを制御ノードへ割り当てて、各機能ブロックの動的挙動を statechart 図²⁾として記述した後、各制御ノードにおいて statechart 図同士を合成する事により、制御ノード用ソフトウェアを状態機械として実装できると考えられる。

図 1 に示すように、DCS の動的挙動は、statechart 図で表される各ノードの状態機械と statechart 図間のイベント連鎖で表現される。ゆえに、DCS の開発には、各ノードの状態機械を実装する手法の開発が必要である。本研究では、statechart 図で記述された制御ノードの状態遷移仕様を Java コードとして実装した後、状態機械間のイベント連鎖を定義する DCS シミュレーションモデル開発デザインパターン²⁾³⁾を提案してきた。現在、DCS のシミュレーションと DCS の実装において、制御ノードの状態遷移仕様を再利用し、かつ制御ノード用ソフトウェアを効率的に開発するために、これらのパターンから得られたオブジェクト記述を制御ノード用ソフトウェアへ変換する手法が求められている。

そこで本報では、我々が提案してきたパターン²⁾³⁾から制御ノードの状態遷移仕様の形式記述へ変換した後、それを LonWorks ネットワークにおける制御ノード用ソフトウェア開発言語である Neuron C で記述された実行可能コードへ変換する手法を提案する。さらに本報では、この手法を実行するソフトウェアとして Statechart コンパイラを開発する。

2. Statechart パターンと Event-Chain パターン

これまで我々が提案してきたデザインパターンのうち、Statechart パターン²⁾は statechart 図で記述された制御ノードの状態遷移仕様を状態機械、状態、状態遷移オブジェクトからなる Java コードとして実装するパターンである。これらのオブジェクト記述は、状態機械を持つクラス（制御ノードとそれに接続されたセンサ・アクチュエータのクラス、図 1 参照）のコンストラクタへ実装される。一方、Event-Chain パターン³⁾は、状態機械のイベントと他の状態機械のアクションの間をバインディング情報として連鎖的に接続するパターンである。これらのイベントとアクションは、入出力変数の値の更新通知と更新処理をそれぞれ記述している。

3. 制御ノードの状態遷移仕様の形式記述

2 節で述べた Statechart パターンのオブジェクト記述は、制御ノードの状態遷移仕様を全て含んでいる。また、制御ノードの状態遷移仕様は、制御ノード自身の機能的な状態

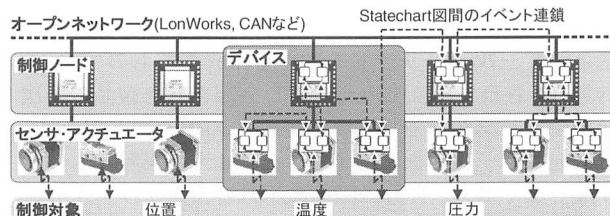


図 1. 状態機械とイベント連鎖を持つ DCS の構造

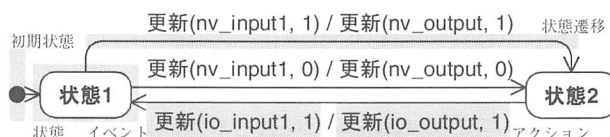


図 2. statechart 図の具体例

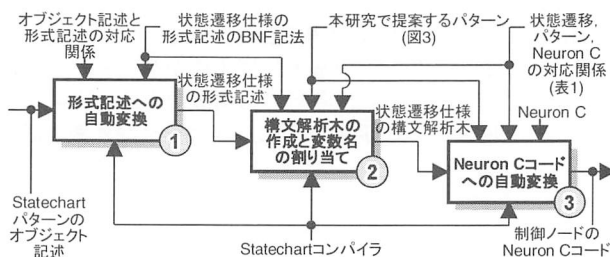


図 3. 本研究で提案する制御ノード用 Neuron C コード開発機械とセンサ・アクチュエータの状態機械からなる。これらは一般的に statechart 図のような階層的な状態機械として記述される。ここで、statechart 図の具体例を図 2 に示す。図 2 は階層的な状態機械ではないものの、状態、状態遷移、イベント、アクションを用いて、制御ノードの動的挙動を表現している。図 2 において、io_input1 と io_output1 は I/O オブジェクトを表し、nv_input1 と nv_output1 はネットワーク変数を表している。本研究では、図 3 に示すように、Neuron C のようなクラス概念を持たない言語による実装も考慮して、Statechart パターンのオブジェクト記述を中間言語となる形式記述へ変換した後 (①)、その形式記述を Neuron C といった制御ノード固有の言語へ変換する (②)、③) 手法を提案する。本研究では、BNF 記法を用いて制御ノードの状態遷移仕様の形式記述を以下のように規定する。

```
<statecharts> ::= <state machine list>
<state machine list> ::= <state machine> *
<state machine> ::= 'StateMachine' <name> '{'
  <state list> <transition list> '}'
<state list> ::= (<state> | <state machine>) *
<state> ::= 'State' <name>
<transition list> ::= <transition> *
```

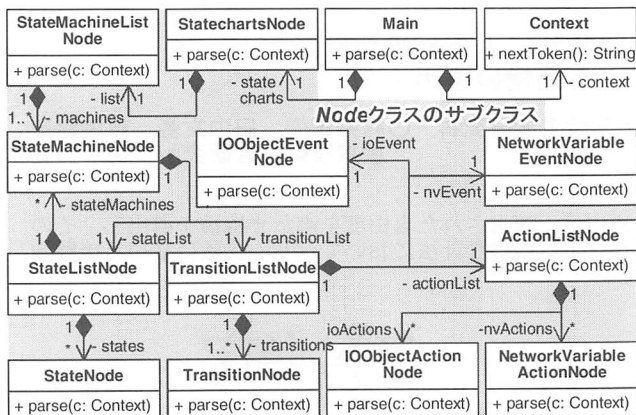


図 4. 状態遷移仕様の形式記述から Neuron C コードへ変換するためのデザインパターン

```

<transition> ::= 'Transition' <name> 'from' <name>
  'to' <name> '{' <event> <action list> '}'
<event> ::= <io object event> |
  <network variable event>
<io object event> ::= 'Event I/O' <number> <name>
  'from' <value> 'to' <value>
<network variable event> ::= 'Event Network' <name>
  'from' <value> 'to' <value>
<action list> ::= (<io object action> |
  <network variable action>)*
<io object action> ::= 'Action I/O' <number> <name>
  'to' <value>
<network variable action> ::= 'Action Network'
  <name> 'to' <value>
  
```

Neuron C は、I/O オブジェクトとネットワーク変数の値の変化をイベントとする ANSI C のサブセットであり、クラス概念がなく、実装メモリも 64K バイト未満であるため、図 3 のように形式記述から Neuron C コードへ変換するためには、構造体と関数を用いて、省メモリで、かつ Statechart パターンと同等の機能を持つ状態機械を実装できるようなデザインパターンが必要とされる。

4. デザインパターンを用いた状態遷移仕様の形式記述から Neuron C コードへの自動変換手法

3 節で議論された要求条件を考慮して、本研究では、図 4 に示すデザインパターンを提案する。このパターンは Gamma らが提案した Interpreter パターン⁴⁾を一部拡張している。図 4 において、網掛け部分の上位クラス Node は抽象クラスであり、構文解析木のノードを表す。また、Node の各サブクラスは、3 節で述べた BNF 記法の各左辺と 1 対 1 の対応関係を持つ。

ここで、図 4 のパターンを用いて図 3 における第 2、第 3 のプロセスを実行する事によって、制御ノードの状態遷移仕様の形式記述から Neuron C コードへ自動変換する手法の手続きを以下に述べる。

- (1) Main オブジェクトの parse() メソッドを呼び出すと、状態遷移仕様の形式記述ファイル (.scd ファイル) が読み込まれて、Statecharts オブジェクトをルートとする構文解析木が作成される。
- (2) Main オブジェクトが Statecharts オブジェクトの別のメソッドを呼び出すと、Statecharts オブジェクトから順番に構文解析木をトレースしながら、各オブジェクトに対して、Neuron C コードで使用される変数名が新規に定義され、そのオブジェクトへ割り当てられる。
- (3) Neuron C ソースファイル (.nc) を作成した後、再び構文解析木を追跡しながら、各オブジェクトへ割り当てられた変数名から Neuron C コードが作成され、そのソース

表 1. 状態遷移仕様、デザインパターン、Neuron C コードの対応関係

状態遷移仕様	デザインパターン	Neuron C コード
状態	StateNode オブジェクト	State 構造体の変数
状態機械	StateMachineNode オブジェクト	StateMachine 構造体の変数
状態遷移	TransitionNode オブジェクト	Transition 構造体の変数
イベント	EventNode オブジェクト	Event 構造体の変数、 入力関数の呼出
アクション	ActionListNode オブジェクト	出力関数の呼出

```

#include <snvt Lev.h>
#include "statecht.h"
void initializeStateMachines();
void eventOccurred(
  StateMachine*, Event*);
struct StateMachine sml[1];
struct State s1[2];
struct Transition t1[2], t2[1];
struct Event e1[3];
network input SNVT lev_disc
  nv_input1 = ST_OFF;
network output SNVT lev_disc
  nv_output1 = ST_OFF;
IO_0 input bit io_input1 = 0;
IO_1 output bit io_output1 = 0;
void initializeStateMachines()
  when(reset) {
    initializeStateMachines();
  }
  when(nv_update_occurs(
    nv_input1)) {
    if(nv_input1 == ST_ON) {
      nv_output1 = ST_ON;
      eventOccurred(&sml[0], &e1[0]);
    } else {
      nv_output1 = ST_OFF;
      eventOccurred(&sml[0], &e1[1]);
    }
  }
  when(io_changes(io_input1) to
    1) {
    io_out(io_output, 1);
    eventOccurred(&sml[0],
      &e1[2]);
  }
  }
  
```

図 5. Statechart コンパイラから出力された Neuron C コードの具体例

ファイルへ書き込まれる。図 4 のパターンにおいて、状態遷移仕様、図 4 のデザインパターン、Neuron C コードの対応関係は、表 1 のように明確に定義されている。

5. Statechart コンパイラ

4 節で述べた自動変換手法を実行するソフトウェアとして、本研究では Statechart コンパイラを開発した。Statechart コンパイラは図 4 と同一の構造を持ち、3 節で述べた状態遷移仕様の形式記述を Neuron C コードへ変換する事ができる。Statechart コンパイラが出力した Neuron C コードの例を図 5 に示す。図 5 において、initializeStateMachines() 関数の手続きも自動生成される。

6. まとめと今後の課題

本報では、Statechart パターンのオブジェクト記述を制御ノードの状態遷移仕様の形式記述へ変換した後、それを実行可能 Neuron C コードへ変換する手法を提案した。さらに本報では、この手法を実現するソフトウェアとして、Java 言語を用いて Statechart コンパイラを開発した。

今後は、ガード条件、状態内のアクションなどの複雑な動的挙動もサポートした状態遷移仕様の形式記述とデザインパターンを提案する予定である。

参考文献

- [1] Object Management Group : OMG Unified Modeling Language Specification Version 1.3, 2000.
- [2] 戸村, 金井, 岸浪, 上広, 山元 : 超分散システム制御ネットワークシミュレータの開発 (第 2 報) - 状態遷移仕様のデザインパターンを用いた制御対象モデルの迅速開発 - , 2000 年度精密工学会春季大会学術講演会講演論文集, p. 71, 2000.
- [3] 戸村, 金井, 岸浪, 上広, 山元 : UML とデザインパターンによる分散制御シミュレーションモデルの設計と実装, 情報処理学会第 62 回全国大会 (平成 13 年前期) 講演論文集 CD-ROM, 2001.
- [4] 結城 浩 : Java 言語で学ぶデザインパターン入門, ソフトバンク, 2001.