5F－4　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 — イベント連鎖からのDCSネットワーク構成情報の抽出・実装 —*

戸村 豊明†
旭川工業高等専門学校‡
金井 理　岸浪 建史††
北海道大学工学部§
伊深 和浩　上広 清　山元 進†††
モトローラ株式会社¶

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）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（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，各機能ブロックをstatechart図[1]として記述した後，各制御ノードでstatechart図同士を合成する事で，制御ノード用ソフトウェアを状態機械として実装可能と考えられる．

図1のように，DCSの挙動はstatechart図で表される各制御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方法論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実行可能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するために，これらのデザインパターンから得られたオブジェクト記述を実行可能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における制御ノード用ソフトウェアの開発言語である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，Java言語で書かれたStatechartコンパイラを開発する．

2. Statechartパターン，Event-Chainパターン

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）のコンストラクタへ実装される．一方，Event-Chainパターン[3)]は，

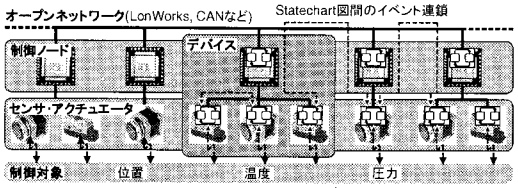

図1. DCSの構造

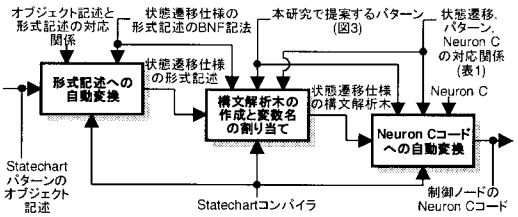

図2. デザインパターンを用いた制御ノード用Neuron Cコード開発手法

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，入出力変数値の更新通知と更新処理を記述している．

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．本研究では図2に示すように，Neuron Cのようなクラスの概念を持たない言語での実装も考慮して，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，その形式記述をNeuron Cといった固有の言語へ変換する手法を提案する．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> *
<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>
```

* An integration of simulating and implementing DCS by design patterns — extracting and implementing configuration information of DCS network from event chain —

† Toyoaki Tomura
‡ Asahikawa National College of Technology
†† Satoshi Kanai, Takeshi Kishinami
§ Faculty of Engineering, Hokkaido University
††† Kazuhiro Ibuka, Kiyoshi Uehiro, Susumu Yamamoto
¶ Motorola Japan

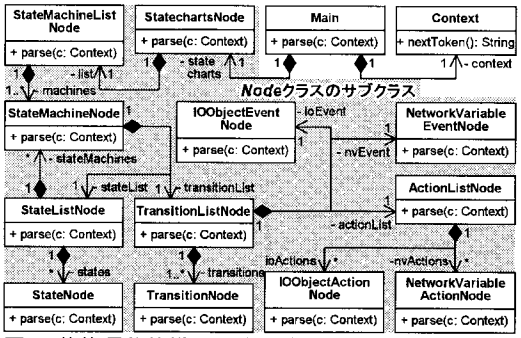

図3. 状態遷移仕様の形式記述からNeuron Cコードへ自動変換するためのデザインパターン

```
<network variable event> ::= 'Event Network' <name>
   'from' <value> 'to' <value>
<action list> ::= (<io object action> |
   <netwrok 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値の変化をイベントとするC言語で，クラスの概念がなく，実装メモリも64Kバイト未満であるため，図2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，簡潔に状態機械を実装できる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

3節で議論された要求条件を考慮して，本研究では，図3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3において，網掛け部の上位クラス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記法の各左辺に1対1対応する．

ここで，図3のパターンを用いて，図2の第2，第3のプロセスを実行する事により，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，割り当てられる．

(3) Neuron C ソースファイル（.nc ファイル）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，そのソースファイルへ書き込まれる．図3のパターンにおいて，状態遷移仕様，

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	ActionList オブジェクト	出力関数の呼出

```
#include <snvt_lev.h>
#include "statecht.h"
void initializeStateMachines();
// 実装済みの関数
void eventOccurred(
  StateMachine*, Event*);
struct StateMachine sm1[1];
struct State s1[3];
struct Transition t1[1], t2[1];
struct Event e1[2];
network input SNVT_lev_disc
  nv_in1 = ST_OFF;
network output SNVT_lev_disc
  nv_out1 = ST_OFF;
when(reset)
{
  initializeStateMachines();
}
when(nv_update_occurs(nv_in1))
{
  nv_out1=(nv_in1 == ST_ON)
    ? ST_ON : ST_OFF;
  eventOccurred(&sm[0],
    &e1[0]);
}
void initializeStateMachines()
{
  // 構造体変数のメンバ値を設定.
}
```

図4. Neuron Cコードの具体例

図3のデザインパターン，Neuron Cコードの対応関係は，表1に示す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，Java言語で書かれたStatechartコンパイラを開発した．Statechartコンパイラは図3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4に示す．図4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，この変換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Neuron Cコード用のデザインパターンを提案してゆく予定である．

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.