第10章

イベント駆動システムの設計と実装を体験する

カナダQNX Software Systems社のリアルタイムOS「QNX」

岡澤幸一

ここでは，リアルタイムOSを利用するイベント駆動システムの設計と実装の方法について解説する．イベントに基づいて動作するシステム(例えば，搬送制御システムなど)を従来のフロー型のソフトウェア設計手法で開発すると，不つごうが生じやすい．前半ではイベント駆動システムの考えかたについて紹介し，後半ではリアルタイムOSを使用した場合の設計や実装の手順とサンプル・プログラムの実行例を示す．ここで使用するリアルタイムOSはカナダQNX Software Systems社の「QNX」である．（編集部）

コンピュータ技術者は，コンピュータがこの世に登場してから現在まで，システムをコンピュータで実現するために，システム設計からプログラム・コードとして動作する実装までを繰り返し行ってきました．当初は1本のプログラムのフローを細かくすることによって，設計を行っていました．しかし，現在では複雑な制御を記述するため，プログラム間の連動や同期，タイミングといった面も考慮しながら設計する必要があります．

また，システムの規模が大きくなると，サブシステムに分解していくことが求められます．この場合，実体としてのコンピュータとその接合点のインターフェースに分離されますが，システムを分解していくソフトウェア的なサブシステムとハードウェアの構成から決まるサブシステムが，必ずしも1対1の対応にならないという問題が起こります．

本チュートリアルでは，こうした問題を解決するイベント駆動システムの基礎について説明します．また，イベント駆動の考えかたに従って記述した搬送制御システムのサンプル記述を，実際のツールを使って実行してみます．ここで利用するツールは，カナダQNX Software Systems社のリアルタイムOS「QNX」と，その開発環境「QNXリアルタイムプラットフォーム」です．

●システム開発の従来の流れ

UML（Unified Modeling Language）などのツールが注目を集めているのは，従来，システムを設計する際に経験的に持っていた方法論が，統一的な手法と共通の検証ツールとして提供される点にあると思います(p.134のコラム「イベント駆動システムの弱点を補完するUMLとSpecC」を参照)．従来型のソフトウェア開発現場において制御システムを開発する場合，おおむね次のような工程をたどっていました．

1）**システム概念設計**――システムに関わるイベントをすべて抽出して，動作を設計する工程です．この工程でシステムとしての故障や異常も把握しておきます．ここではサブシステムなどへの分解は行いません．

2）**システム外部設計**――システム概念設計で作られたシステムの動きを実際のコンポーネントとして設計し，サブシステムの統合としてシステムを設計する工程です．システム概念設計で作成したシステムの流れをサブシステム間のインターフェースとして分解します．

3）**システム実装**――システム外部設計で作られた仕様を実際のプログラムという形に展開する工程です．これまでは，内部設計，内部仕様の決定，テスト設計という手順でプログラムの開発を行い，コーディングを通してコンポーネント（ソフトウェア部品）を作っていました．

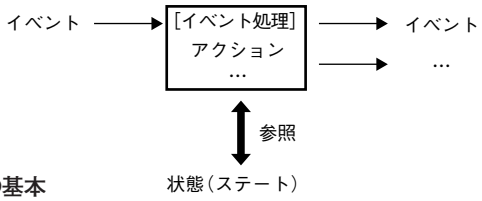
こうした開発フローをイベントで動作するシステムの開

発に適用した場合，多くの点で不整合が起きることを経験します．例えば，以下のような不整合です．

- 概念設計に変更があったとき，システム全体の実装を大規模に変更する必要がある．場合によっては，全面的な作り直しになることもありうる．
- イベントの動きに対応させて設計しても，動作コンポーネントはイベントに対応しにくいワン・フローのプログラム（main()…）で記述しなければならない．
- 大規模なシステムを構築すると，サブシステム間のインターフェースが肥大化して，そのテストだけで長い期間を費やしてしまう．
- 設計時の資料が紙であるため，イベント・フローをチェックするときに，場合によっては壁一面に設計図を張り出してイベントをトレースするはめになる．

1．イベント駆動システムの構成要素

ここで，「リアルタイム・システムはイベント駆動で動作する」と定義すると，ずいぶんと見通しがよくなります．筆者もフロー型でプログラムを記述するということに慣れきっていたため，イベントという考えかたを導入したとき，それを理解し，設計から実装までを身につけるまでずいぶんと時間がかかりました．イベントには制御に特有のデバイス・イベントや，GUIに代表されるユーザ・インターフェースに関わるユーザ・イベントが存在します．二つの違いは反応時間のオーダです．デバイスはμsオーダの反応時間を要求しますが，人間は数十msオーダであれば許容します．



〔図1〕
イベント動作の基本

●イベント，アクション，状態から成るイベント駆動

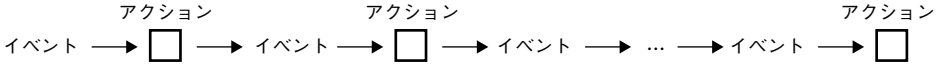
イベントで動作するシステムの基本は，**図1**のように表現できます．イベントに対応して，複数のアクションからなるイベント処理が，状態（ステート）を参照しながら動作します．アクションの中には状態の変更やイベントの新たな発生も含まれます．

さらに，システム設計の視点で見るとイベントのつながりをシーケンスとしてとらえ，**図2**のようなイベント・シーケンス図として扱えます．テスト段階では，このシーケンスが設計したとおりに流れるかどうかが問題となります．

●処理時間の制約が厳しいリアルタイム・システム

リアルタイム・システム（実時間システム）の場合，イ

▶〔図2〕イベント・シーケンス



COLUMN　イベント駆動システムの弱点を補完するUMLとSpecC

イベント駆動システムを扱ううえで，二つの課題があります．それは設計手法と実装における記述言語です．

設計手法については，これまで現場の経験の中で方法論が築かれてきましたが，全体にわたって統一された手法は存在しませんでした．もちろん状態遷移による設計手法などはありました．その中で，UML（Unified Modeling Language）による方法論のように，「オブジェクト，状態モデル，シーケンス図」を統一的に扱って設計する手法に，いままでの経験と照らし合わせて，筆者は美しさと大きな可能性を感じています．ただし，UMLによる方法論は実装に展開する手法が弱いという点が残念です．UMLツールの最終点は，現状ではC++に展開してシミュレーションするレベルです．UMLが実用に耐えうるリアルタイムOSの環境とリンクすることを期待していただけに，筆者にとって，UMLツールの設計支援ツールという位置付けを残念に思っています．

実装については，CやC++では，スレッドの並列性や同期を関数としてしか記述できません．実は筆者はVerilog HDLが持つ並列処理の記述性に以前から注目していました．現在では，この点についてSpecCに期待しています．SpecCはハードウェアの論理回路に存在する並列性とタイミングを記述できます．このことは，イベント駆動システムのソフトウェアを記述する上でも，大きな役割を果たすのではないかと，期待することしきりです．