B-009

状態遷移図を用いたリソース管理設計
Design Tool for Inter-Application Resource Management

塩本　佳子†
Yoshiko Shiomoto

1. はじめに

複数のアプリケーションで構成されるシステムでは、デバイスなどの限られたリソースを共有するために、リソースの管理が重要な問題となる。一方、変更・拡張が容易なシステムにするためには、アプリケーション間の依存性を極力排除する必要がある。そのためには、アプリケーション間の調整を行うマネージャを導入し、個々のアプリケーションがマネージャとのみ交渉するデザインパターンの適用が、1 つの有効な手段である。また、固定的な処理と、仕様に応じて変化する処理を分離させることで、システムの変更・拡張の影響範囲を更に限定することが出来る。我々は、仕様に応じて変化する部分を、リソースの割り当て条件と、割り当てに伴う処理であるととらえ、この部分をデータ化するとともに、データを作成するための設計ツールの検討・開発を行った。

2. ツールの概要

2.1. 状態遷移図を使用した記述

リソースの割り当て条件とは、あるアプリケーションがリソースを使用している状態において、新たなリソース要求が発生した場合の動作（どのアプリケーションにリソース使用権を与えるか）を定義したものである。これは、あるアプリケーションがリソースを使用している状態でのリソース要求イベントという、1 対 1 の動作の設定であり、ある種の状態遷移図として記述できる。

ここで想定している状態遷移図は、複数の「状態」が存在し、「イベント」をトリガとして「動作」を伴い他の「状態」に遷移する過程を、有向グラフとして記述するものである。状態間の矢印で遷移を記述し、矢印の属性として、トリガとなるイベントと、このイベントによって引き起こされる動作を設定する。イベントにはシステムで発生し、マネージャが受け取るイベントを、動作にはマネージャやアプリケーションが持つ機能を指定する。「状態 A」においてイベント 1 が発生した場合に、動作 1 を引き起こして「状態 B」に遷移する例を図 1に示す。

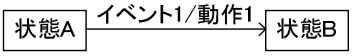

図 1　使用する状態遷移図の形式

設計ツールでは、状態遷移図でリソースの割り当て条件を記述することで、リソース割り当てに関する動作仕様を設計し、データを作成する。各「状態」がアプリケーションに対応し、ある「状態」がアクティブである時、この「状態」に対応するアプリケーションがリソースを占有するものとする。なお、ここでは、常に 1 つのアプリケーションがリソースを占有する場合を対象とし、並行状態は扱わない。

2.2. 記述上の特徴

状態遷移図を用いた記述では、状態数の増加による可読性の低下を避けるため、複数の状態をグループとして一括した記述を可能にする。グループに設定した定義は、内包する全ての「状態」に適用する。

グループに記述した条件と異なる条件を内部で再定義した場合には、内側の定義を優先する。これは、『ほぼ似たような動作を行う』状態をグループ化できるようにし、システム全体の大きな遷移の把握を容易にするためである。図 2にグループを使用した記述の例を示す。この例では、状態 A,B,C はイベント 2 に対しては一部動作が異なるが、イベント 4 に対しては同じ動作（状態 E 遷移）を行うことが分かる。

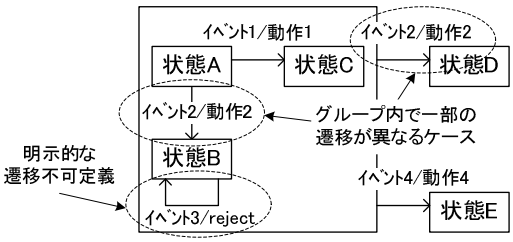

図 2　グループを使った記述

また、状態遷移図では、遷移として記述されない組み合わせを遷移不可として扱うが、グループ内での例外動作として遷移不可を定義するために、遷移不可を自己遷移として明示的に記述可能とする（図 2. 状態 B におけるイベント 3 の動作）。

3. リソース管理への適用

3.1. 単純なリソース管理

リソース要求順に単純な排他制御のみを行う場合は、マネージャは基本的に状態遷移図に従った動作を行えばよい。すなわち、マネージャはリソース要求イベントを受け取って、状態遷移図における現在の「状態」に対する遷移を確認する。遷移が定義されていれば、指定された遷移先の「状態」に遷移し、対応するアプリケーションにリソース使用権を移動させ、遷移が定義されていなければリソース使用権を移動させない。

3.2. 遷移履歴を考慮したリソース管理

リソース管理にリソース要求イベント以外の条件も考慮する場合、状態遷移図だけで全てを記述すると、記述が複雑になる場合がある。このような場合は、マネージャとの協調動作を導入して、状態遷移図の簡素化を図る。ここでは、遷移履歴の考慮が必要なシステムを例に、状態遷移図の簡素化の方法を示す。

例えば、3 つのアプリケーション A,B,C からなるシステムを考える。ここで、アプリ A,B は、アプリ C のリソー

† 三菱電機(株)先端技術総合研究所

ス要求に対しては、一時的にリソースの使用を中断してリソースの使用権を譲り、アプリ C の終了後にリソースの使用を再開するものとする。また、アプリ A は、アプリ B のリソース要求に対しては、リソースの使用を終了してリソースの使用権を譲るものとする。

このようなリソース管理を状態遷移図のみで記述した例を図 3に示す。遷移のガード条件として遷移元を指定する（図 3-a）、遷移元毎に状態を分けて管理する（図 3-b）、などの方法が考えられるが、いずれも遷移履歴を明示的に扱う必要があり、状態遷移図の記述が煩雑なものとなる。

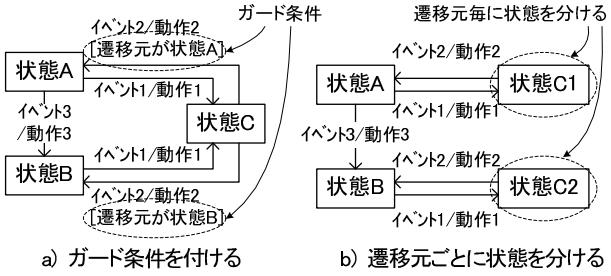

図 3　遷移元を考慮した記述の状態遷移図

そこで、状態遷移図から遷移履歴に関する記述を分離するため、マネージャがスタックを利用して遷移履歴を管理することとする。

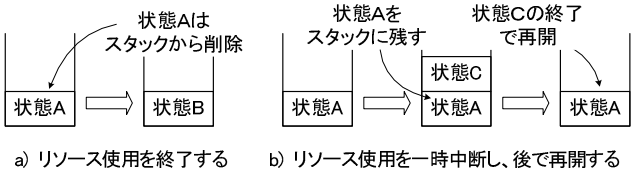

図 4　スタックを使用した戻り先の管理

新たなアプリケーションがリソースの使用を開始する際は、マネージャが対応する「状態」をスタックに追加する。このとき、前のアプリケーションがリソースの使用を終了する場合は、スタックから削除する（図 4-a）。一方、前のアプリケーションがリソースの使用を一時的に中断する場合は、対応する「状態」をスタックに残しておく（図 4- b 左）。一方、新たなアプリケーションがリソースの使用を終了すると、対応する「状態」をスタックから削除し、下位の「状態」に対応するアプリケーションにリソースの使用権が移動する（図 4 - b 右）。

一方、状態遷移図では、「動作」としてマネージャでのスタック操作を指定する。スタック操作は、change（現在の状態をスタックから削除して遷移先の状態をスタックに積む）、push（現在の状態をスタックに残したまま、遷移先の状態をスタックに積む）、pop（現在の状態をスタックから削除する）の 3 種類を考える。図 3の例で「動作」にスタック操作を指定した状態遷移図を図 5に示す。

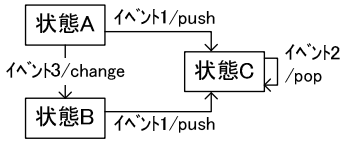

図 5　スタック操作を指定した記述

この例では、「状態 A」から「状態 C」に遷移した場合は、push 操作が指定されているため、「状態 A」はスタックに残る。イベント 2 の発生で「状態 C」がスタックから削除されると、スタック下位に残っていた「状態 A」が戻り先となる。

明示的な状態遷移とはならないが、pop 動作を自己遷移として記述することで、状態遷移図上で遷移元に関する記述が不要になるだけでなく、個々の戻り先への遷移を省略でき、記述が簡素化できる。また、戻り先となる「状態」を明示的に記述しないため、リソース使用を中断しているアプリケーションが待機状態を放棄した場合でも、対応する「状態」をスタックから削除することで、戻り先はもう 1 つ前の「状態」に自動的に変更される。

4. システムへの適用評価

実用レベルのシステムに対し、設計ツールを用いて画面管理の設計を行った。

画面の管理は同じアプリケーションであっても、内部の状態によって動作が異なるため、アプリケーションを複数の「状態」に分け、画面表示要求を「イベント」とした結果、9 つのアプリケーションに対し、状態数・イベント数は 37 となり、組み合わせは 37×37 のマトリックスとなった。この規模では、遷移表などで個々の組み合わせについて個別に動作を定義するのは困難であった。

しかし、状態遷移図を用いた記述では、定義した遷移の数は 105 に留まり、遷移の流れの把握が容易になった。また状態遷移図を使用することで、

- リソースの使用権の移動が視覚的に把握できるため、仕様変更や障害への対応で定義を変える場合でも、全体への影響が把握しやすい。
- 状態遷移図から状態遷移表への変換も簡単に行えるため、設計では状態遷移図を使用し、データやテスト用には表形式で出力するなど、目的に応じたデータが生成できる。
- 例外記述可能なグループの導入による遷移の削減に加え、状態遷移図では記述されない組み合わせが暗黙的に「遷移不可」として扱える。

といったメリットも確認できた。

5. おわりに

今回、状態遷移図を用いてリソース割り当ての動作仕様を記述する設計ツールを作成し、スタックを用いたリソース管理を行うマネージャを対象にリソース管理設計を行った。動作仕様部分をデータ化し、これを設計ツールで設計することにより、開発工程で生じた動作仕様の変更に容易に対応できることが確認できた。状態遷移図として記述した要求仕様から、直接ソースに組み込むデータを生成するため、データ作成の手間や修正漏れなどがなくなった。

今後は、更に複雑なリソース管理条件を持つシステムについて、適用検討を進める予定である。