振る舞いが完全にはわからないモジュールを含むシステムの設計手法

北陸先端科学技術大学院大学
情報科学研究科

高橋　聡樹

平成 30 年 3 月

課題研究報告書

振る舞いが完全にはわからないモジュールを含むシステムの設計手法

1410753　　高橋　聡樹

主指導教員　　青木　利晃

審査委員主査　　青木　利晃

審査委員　　平石　邦彦

田中　清史

北陸先端科学技術大学院大学

情報科学研究科

平成 30 年 2 月

目次

1. はじめに

1.1. 背景

近年システムの重要性が増してきており、システムが担う機能も複雑になっている。それに伴いシステム開発も大規模で複雑になってきている。そのためシステム設計時には複雑な機能を実現させるために、機能を複数のモジュールで分割し、モジュール間のメッセージ送受信を通じ、ユーザーの要求を実現することがある。またメッセージの送受信の中でモジュールの状態が変わる場合は、それを状態遷移モデルとして設計する。

モジュール同士が非同期で並行協調動作しながらメッセージの送受信をする場合、ソフトウェアがとり得る状態が多くなり、設計を難しくさせることがある。特にモジュールが外部から提供されている場合、モジュール内の詳細な振る舞いがわかり難い。例えば商用で提供されるモジュールは仕様書という形でインタフェースは公開されるが、ソースコードまでは公開されず、モジュールがブラックボックスであることがある。このときインタフェース仕様書からモジュールの振る舞いをモデル化する必要がある。しかし、仕様書は自然言語で記述されている部分があり、自然言語の記述の曖昧さからモデルの制約を正確に把握することが難しいことがある。

1.2. 研究動機

ここでは筆者が本研究を行おうと考えた動機について記述する。筆者は過去にシステムエンジニアとして車載オーディオシステムを開発するプロジェクトに参加していた。開発するオーディオシステムは複数の並行協調動作するモジュールによって構成されており、モジュールは非同期で送受信されるメッセージを通じて動作した。モジュールの 1 つは他社製のモジュールを使い、システムの 1 部として組み込まれていた。筆者はシステムのモジュールの 1 つの設計を行っており、システムのユースケースを想定しながらモジュール間のメッセージのやり取りをシーケンス図として作成したり、モジュールの状態遷移設計を担当した。設計業務を行う中で感じたのは、他社製モジュールのような外部から提供され、内部の構造がわからないブラックボックスなモジュールと並行協調動作するシステムの開発は難しいということでした。通常このようなブラックボックスなモジュールは自然言語のインタフェース仕様書が提供され、仕様書からモジュールの振る舞いを想定する。しかし、仕様書は自然言語で書かれているため、自然言語の曖昧さから必ずしもモジュールの振る舞いを把握できるわけではない。そしてこのような振る舞いを把握し難いモジュールと並行協調動作するシステムに対し、ユースケースを想定しながら振る舞いパターンを洗い出すのが非常に難しい。モジュールそれぞれが状態を持ち、モジュール間をやり取りするメッセージが多くなると、全てのパターンを設計者は把握しきれず、それによって設計不具合を出してしまう。こういった設計時の問題を解決したいと思い、この研究を行おうと考えた。

既存の研究では誤りを検出するものとしてソースコードから誤りを特定したり[10]、シーケンス図のような設計図から誤りを検出する手法が存在する[13, 14, 16]。しかしこれらは

何れも検証対象となるシステムの実装や設計がわかっている前提での手法であり、振る舞いが完全にわからないようなブラックボックスなモジュールを含んだシステムには適用しにくい。そのため振る舞いが完全にはわからないようなブラックボックスなモジュールを含んだシステムを検証できる手法が求められる。

1.3. 研究目的

本研究では次の要件を満たすような設計手法を提案する。

1. 設計者の MW の想定外の振る舞いに起因する不具合を見つけることができること
2. 手法が半形式的に行えること
3. 実際の開発現場に適応できること

1 はもともとの研究動機となった不具合と同じ状況の不具合であり、実際の開発現場でもよくあるため目的として設定した。2 は手法を手順化し、不具合を見つけることができるようになる必要があるためである。3 は実際の開発現場でも適応できるような手順の複雑さや、方法である必要はあるためである。

1.4. 提案手法について

本研究では複数のモジュールが並行して協調動作するソフトウェアのうち、仕様書からのみ振る舞いが把握できるブラックボックスなモジュールを含むソフトウェアを検証する手法を提案する。提案手法ではインタフェース仕様書の一般系について定義する。次にインタフェース仕様書の制約から状態遷移モデルを作成する方法を示す。インタフェース仕様書には自然言語が含まれるため、モデル作成の曖昧さが含まれる問題がある。そこで本手法ではモデルを作成する際に実際には起きえない振る舞いも含めて状態遷移モデルに反映させる。

システムが含む全てのモジュールをモデル化したとしてもモデルがとり得るパターンが多くなることがある。そこで、作成したモデルに対して満たすべき性質でモデル検査を実施し、反例がないか検査する。反例が出た場合、モデルを修正する必要があるが、これがブラックボックスなモジュールのモデルに関する起きえない振る舞いに起因する場合は起きえない振る舞いを排除するようにモデルを修正する。それ以外のモジュールのモデルが起因する場合はそれが設計不具合であるため、修正するようにする。これによりブラックボックスなモジュールについてもモデル検査を適応でき、ソフトウェアの品質を向上させることができる。

2. 関連技術

2.1. モデル検査

モデル検査は形式手法の 1 つであり、ソフトウェアの信頼性向上のために用いられる技術である。システムの振る舞いをモデルとして記述し、システムの満たすべき性質が成立す

るかどうかを網羅的に探索して機械的に検査する手法である。

2.2. Spin

Spin は AT&T ベル研究所の G.J. Holzmann によって提案されたモデル検査のツールである。Spin はもともと通信プロトコルの検証を目的としたものであり、仕様記述言語である PROMELA を用いて検証対象の振る舞いを記述する。PROMELA では複数のプロセスがチャネルを通してメッセージ通信する振る舞いを記述することができる。Spin ではラベル、表明、性質オートマトンで指定した性質を検査できるほか、線形時相論理式(LTL)を用いてより詳細な性質を検査することができる。検査した結果、PROMELA で記述した振る舞いが満たすべき性質を満たさない場合は、性質に反する実行例を反例として示す。

2.3. システムとモジュール

システムが機能ごとに分割され、それらが平行動作するとき、本論文では分割された機能の単位をモジュールと呼ぶ。

2.4. 状態遷移モデル

システムの振る舞いを記述する方法に状態遷移モデルがある。状態遷移モデルとは各状態を頂点とし、頂点を結ぶ辺を状態間の遷移としてグラフィカルに表現する記法である。状態遷移モデルの記法や呼び方は用途に応じていくつか存在するが、本研究では次のように定義する。

図 2.4-1 は「遷移元状態」、「遷移先状態」、「トリガー」、「アクション」の関係を表したものである。図 2.4-1 にある線で結ばれる四角の形状が状態である。状態間は方向を持つ矢印線で結ばれる。図 2.4-1 のように状態 A から状態 B の向きへ線が引かれるとき、状態 A は状態 B へ遷移することを表している。遷移時に条件となる動作がある場合、この条件を「トリガー」と呼び、遷移の矢印線の上に記述する。遷移時に何らかのシステムの動作がある場合、トリガーの右側に記述し、これを「アクション」と呼ぶ。また 1 つの矢印の遷移に着目した際に、矢印の元の状態を「遷移元状態」、矢印の先の状態を「遷移先状態」と呼ぶ。

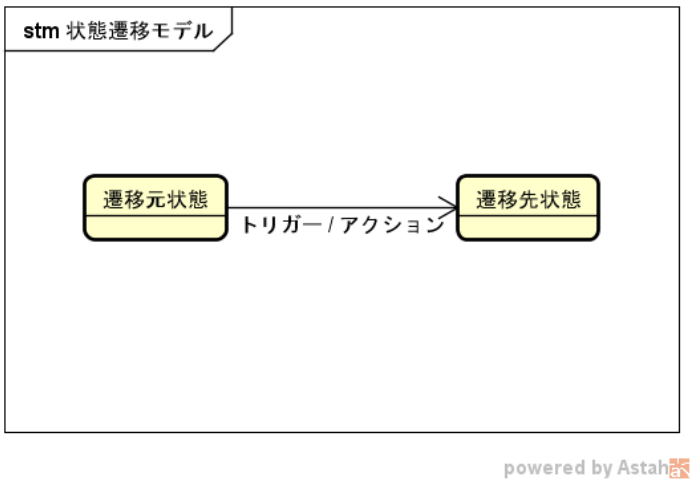

図 2.4-1 状態遷移モデルの例

2.5. インタフェース仕様書

インタフェース仕様書とは振る舞いがブラックボックスなモジュールのインタフェース部分や外部から見たときの振る舞いについて自然言語で記述したものであり、設計者はこの仕様書の記述をもとに結合して動作するモジュールを設計する。

3. インタフェース仕様書の一般化

3.1. 既存の仕様書を用いた考察

本研究の提案手法では、このインタフェース仕様書の記述から状態遷移モデルを作成する。そのため、インタフェース仕様書にどのような要素が含まれているかを実際に存在する仕様書を参考に一般化した。参考にした仕様書として「ITRON TCP/IP API 仕様 Ver. 1.00.01」[2]（以下参考仕様書と呼ぶ）を用いた。この仕様書は ITRON 仕様 OS 上の TCP/IP プロトコルスタックについての仕様を定義した仕様書である。仕様書の記述をもとに一般化した仕様書に含むべき要素について考察する。

3.1.1. モジュールの API と状態

参考仕様書の「2 章 TCP の API」には API と呼ばれるインタフェースについて記述された項目がある。API にはモジュールを操作するためのインタフェースについて記述されている。一般的にこのようなモジュールを操作するためのインタフェース仕様を定義する記述をするため、一般化した仕様書にも同等の記述が必要である。また、仕様書内には API を呼び出したことによるモジュールの状態遷移と、モジュールがとり得る全ての状態が記述される。参考仕様書の「2.1 概要」に掲載されている「図 1.TCP 通信端点の状態遷移」を以下の図 3.1.1-1 に示す。この図では API を呼び出したことによる API の通信対象（モジ

ュール）の状態遷移を表している。仕様書では対象モジュールの内部実装について記述されることはないが、このようにインタフェースを通じてやり取りした際の外部から見た状態については記述される。よって一般化した仕様書にもモジュールがとり得る全状態を記述する必要がある。

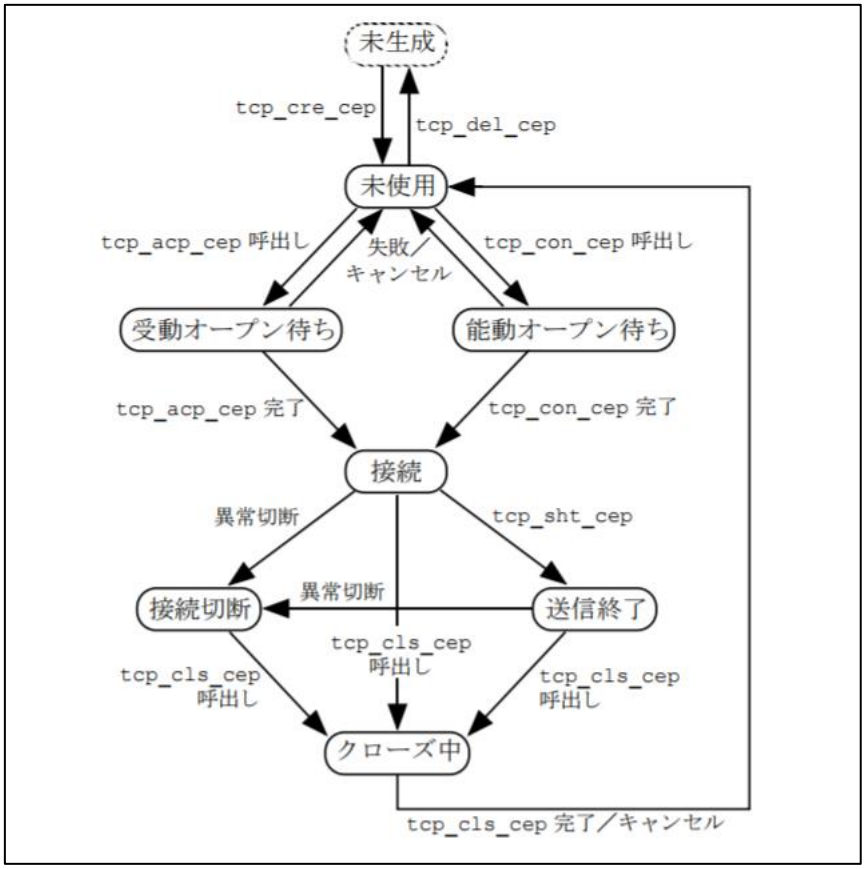

図 3.1.1-1 TCP 通信端点の状態遷移

3.1.2. API 記述

参考仕様書内の API の記述内容を見ながら一般化した仕様書内に必要な記述を考える。図 3.1.2-1 は参考仕様書内にある API の記述例である。この例は参考仕様書内の「2.2 TCP 受付口の生成/削除」に定義されている API の 1 つである。API には【C 言語 API】や【パラメータ】のように項目が設けており、それぞれ API ごとに記述がある。先ず API には「API 名」という形で一意に決まり、それぞれの API を特定する名前が存在する。この例では“tcp_acp_cep”と書かれている部分が該当する。一般的にインタフェースを識別するための名前を記述した項目が存在するので、一般化した仕様書にもこれに相当する項目が必要である。【C 言語 API】と書かれている項目があるが、これは実装依存の項目であり、仕様書ごとに記述の有無が異なるため、一般化した仕様書には盛り込まない。【パラメータ】は API に設定するパラメータを表している。この例では 3 つのパラメータが設定可能である。パラメータには左の列に”ID”や”TMO”記述されている部分があり、これは設定するパラ

メータの型を表している。また、一般的に仕様書内には型に対してどのような設定可能な値が存在するかをが明記されている。この仕様書では「1.5.3 定数」に TMO に設定可能な値が示されている。図 3.1.2-2 に設定可能な値を示している部分を抜粋する。一般的に API には複数校のパラメータが設定されることがあるので、設定可能なパラメータ数が記述される項目が一般化された仕様書には必要である。【リターンパラメータ】、【エラーコード】は呼び出し後に即時でわかる結果について値が設定される項目である。今回の手法では前提条件として非同期で通信する API に限定しており、モデル化もモジュールから非同期で返却される値を対象とする。よって一般化された仕様書の記述にはこの 2 項目は盛り込まない。【API の機能】、【補足説明】は何れも自然言語によって記述される。API についての説明は一般化された仕様書のほとんどに載っているが、記述する内容や項目は仕様書ごとに異なる。そこで一般化された仕様書には API について自然言語で記述される項目を 1 つ盛り込み、自然言語で記述された説明などは全てこの項目に記述されるものとする。

tcp_acp_cep　　接続要求待ち（受動オープン）　　【標準機能】

【C 言語 API】

ER ercd = tcp_acp_cep(ID cepid, ID repid, T_IPV4EP *p_dstaddr, TMO tmout);

【パラメータ】

ID	cepid	TCP 通信端点 ID
ID	repid	TCP 受付口
TMO	tmout	タイムアウト指定

【リターンパラメータ】

T_IPV4EP	dstaddr	相手側の IP アドレスとポート番号
ER	ercd	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー（dstaddr のアドレス，tmout が不正）
E_OBJ	オブジェクト状態エラー（指定した TCP 通信端点が使用中）
E_DLT	接続要求を待っている TCP 受付口が削除された
E_WBLK	ノンブロッキングコール受付け
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	処理のキャンセル，待ち状態の強制解除

【API の機能】指定した TCP 受付口に対する接続要求を待つ．接続要求があった場合には，指定した TCP 通信端点を用いて接続を行い，相手側の IP アドレスとポート番号を返す．接続が完了するまで待ち状態となる．

同じ TCP 受付口に対して，同時に複数の tcp_acp_cep を発行することができる．その場合，接続要求をどの TCP 通信端点に受け付けるかは実装依存である．

tcp_acp_cep を呼び出すことで，TCP 通信端点は受動オープン待ち状態となり，処理完了した時点で接続状態となる．タイムアウトや tcp_can_cep によって tcp_acp_cep の処理がキャンセルされた場合，TCP 通信端点は未使用状態に戻る．

【補足説明】同一の TCP 通信端点に対する tcp_acp_cep がペンディングしている間に tcp_acp_cep を発行した場合，TCP 通信端点が使用中であるために，E_OBJ エラーとなる．

図 3.1.2-1 仕様書内の API 記述例

(4) タイムアウト指定

TMO_POL	0	ポーリング
TMO_FEVR	–1	永久待ち
TMO_NBLK	–2	ノンブロッキングコール

図 3.1.2-2 TMO 型に設定可能な値

3.1.3. コールバック記述

API のコールバック関数について考察する。API ではコールバック関数という形で非同期の要求やデータの通知を受け付けるインタフェースを設けることがある。図 3.1.3-1 は参考仕様書の「2.9 コールバック」の章にあるコールバックについての記述を抜き出したものである。

ノンブロッキングコールの完了通知 【標準機能】

【事象の種類】
終了した API の機能コード

【固有パラメータ】
（以下はパラメータブロックに入れて渡される）

ER	ercd	API からの返値

【返値】
使わない

【コールバックの機能】ノンブロッキングコールの処理が完了した（ないしはキャンセルされた）場合に呼び出される．API からの返値が，パラメータブロックに入れて渡される．API からのその他のリターンパラメータは，API を呼び出した時に指定した領域に格納される．

【補足説明】例えば，ノンブロッキングの tcp_rcv_buf 処理が完了した場合，バッファに連続して入っている受信データの長さが ercd に渡され，バッファの先頭アドレスは tcp_rcv_buf を呼び出した時のパラメータ p_buf が指す領域に格納される．

図 3.1.3-1 仕様書のコールバック記述例

まず、コールバック関数には個別のコールバックを識別するための「名前」が必要である。【事象の種類】と書かれている項目があるが、これはコールバックを識別するためのコードが格納される項目である。コールバック関数を識別する「名前」がすでに一般化した仕様書に盛り込まれるため、この項目は一般化した仕様書には盛り込まない。【固有パラメータ】はコールバック関数に設定するパラメータを表している。コールバックのパラメータも API

と同様に「パラメータの型」と「パラメータ数」が一般化した仕様書に必要である。【返値】はコールバック関数が呼ばれた際の戻り値が設定されるが、今回の仕様書からのモデル化ではモジュール側から通知されたメッセージとしてコールバックをモデル化するため、この項目にモデル化には不要である。よって一般化した仕様書には【返値】は盛り込まない。【コールバックの機能】、【補足説明】は何れも自然言語によって記述される。コールバックについての説明は一般的な仕様書のほとんどに載っているが、記述する内容や項目は仕様書ごとに異なる。そこで一般化された仕様書には API について自然言語で記述される高億を 1 つ盛り込み、自然言語で記述された説明などは全てこの項目に記述されるものとする。

3.2. 一般化した仕様書

3.1 章での考察からブラックボックスなモジュールの一般化した仕様書に含まれる記述要素についてまとめる。インタフェース仕様書には大きく次の 2 つの要素が含まれるとする。1 つはブラックボックスなモジュールがとり得る全ての状態と、もう 1 つはメッセージ仕様である。モジュールがとり得る状態はモジュールがインタフェースのメッセージを通じてやり取りする中で遷移する状態を全て列挙したものである。メッセージ仕様は 3.1 章で考察した API やコールバックが含まれる。

3.2.1. メッセージ仕様

メッセージの仕様の記述方法はメッセージの形式により 3 つのパターンがある。メッセージの形式とはモジュールの呼び出し元とモジュール間のメッセージの方向であり Request 形式、Request-CallBack 形式と Event 形式が存在する。以下にメッセージの仕様の記述について示す。

3.2.2. Request 形式

Request 形式は呼び出し元から呼び出し先のブラックボックスなモジュールへの一方向のメッセージであることを表す。呼び出し側モジュールを App 呼び出される側のモジュールを MW としたとき、モジュール間のメッセージの流れは以下の図 3.2.2-1 のようになる。

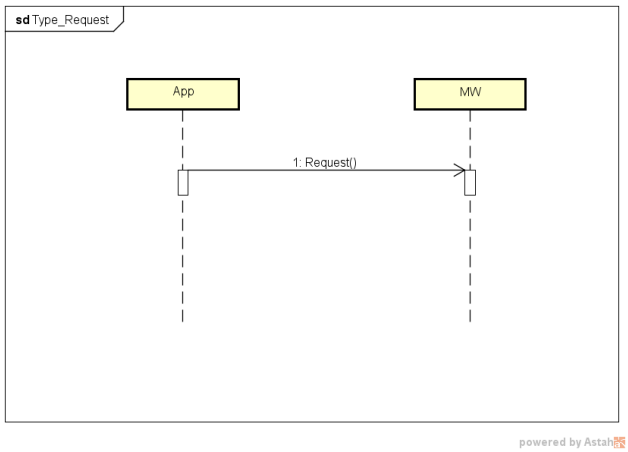

図 3.2.2-1Request 形式メッセージのモジュール間メッセージの流れ

Request 形式のメッセージにはインタフェースの説明に記述される項目としてメッセージ形式、メッセージ名、パラメータ（パラメータ数、パラメータの型)、メッセージの説明が含まれる。説明として記述される項目と、記述される内容の一覧は表 3.2.2-1 のようになる。

表 3.2.2-1 Request 形式メッセージの内容

パラメータ		記述される内容
メッセージ形式		「Request 形式」が固定で記述される
メッセージ名		メッセージ固有の名前
パラメータ	パラメータ数	0 以上の整数
	パラメータの型	パラメータ数が 1 以上の時、パラメータとして設定される値の型が記載される
メッセージの説明		メッセージの使い方や呼び出し時のモジュールの動作が記載される

3.2.3. Request – CallBack 形式

Request-CallBack 形式は呼び出し元から呼び出し先のブラックボックスなモジュールへ送信したメッセージに対しての応答メッセージが非同期で返ってくる形式である。呼び出し側モジュールを App 呼び出される側のモジュールを MW としたとき、モジュール間のメッセージの流れは以下の図 3.2.3-1 のようになる。

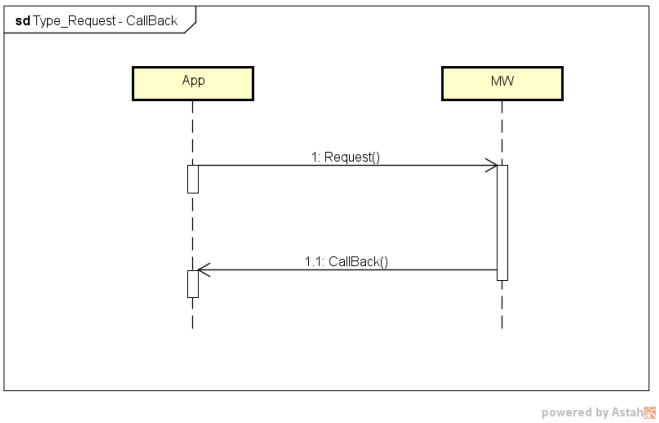

図 3.2.3-1Request-CallBack 形式メッセージのモジュール間メッセージの流れ

Request-CallBack 形式のメッセージにはインタフェースの説明に記述される項目としてメッセージ形式、メッセージ名、パラメータ（パラメータ数、パラメータの型）、メッセージの説明が含まれる。説明として記述される項目と、記述される内容の一覧は表 3.2.3-1 のようになる。また Request-CallBack 形式のメッセージには対応する CallBack 形式のメッセージが必ず存在する。CallBack 形式のメッセージには Request-CallBack 形式のメッセージに対する応答内容が設定される。CallBack 形式のメッセージ説明として記述される項目と、記述される内容の一覧は表 3.2.3-2 のようになる。

表 3.2.3-1 Request-CallBack 形式メッセージの内容

パラメータ		記述される内容
メッセージ形式		「Request-CallBack 形式」が固定で記述される
メッセージ名		メッセージ固有の名前
パラメータ	パラメータ数	0 以上の整数
	パラメータの型	パラメータ数が 1 以上の時、パラメータとして設定される値の型が記載される
メッセージの説明		メッセージの使い方や呼び出し時のモジュールの動作、および対応する CallBack 形式のメッセージがどれであるか記載される

表 3.2.3-2 CallBack 形式メッセージの内容

パラメータ		記述される内容
メッセージ形式		「CallBack 形式」が固定で記述される
メッセージ名		メッセージ固有の名前
パラメータ	パラメータ数	0 以上の整数
	パラメータの型	パラメータ数が 1 以上の時、パラメータとして設定され

		る値の型が記載される
メッセージの説明		メッセージの使い方や呼び出し時のモジュールの動作、および対応する Request-CallBack 形式のメッセージがどれであるか記載される

3.2.4. Event 形式

Event 形式はブラックボックスなモジュール側から一方的に送信される形式のメッセージである。呼び出し側モジュールを App 呼び出される側のモジュールを MW としたとき、モジュール間のメッセージの流れは以下の図 3.2.4-1 のようになる。

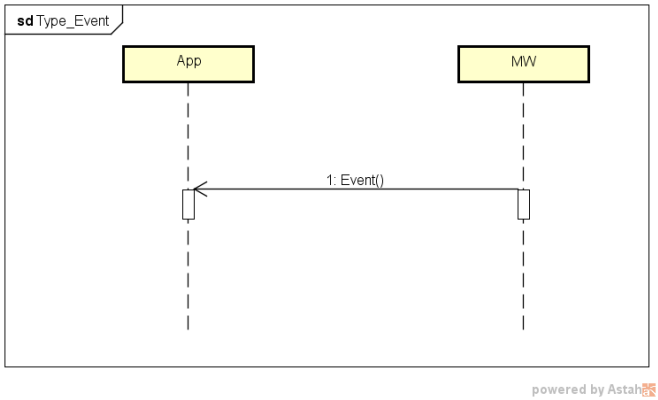

図 3.2.4-1 Event 形式メッセージのモジュール間メッセージの流れ

Event 形式のメッセージにはインタフェースの説明に記述される項目としてメッセージ形式、メッセージ名、パラメータ（パラメータ数、パラメータの型)、メッセージの説明が含まれる。説明として記述される項目と、記述される内容の一覧は表 3.2.4-1 のようになる。

表 3.2.4-1 Event 形式メッセージの内容

パラメータ		記述される内容
メッセージ形式		「Event 形式」が固定で記述される
メッセージ名		メッセージ固有の名前
パラメータ	パラメータ数	0 以上の整数
	パラメータの型	パラメータ数が 1 以上の時､パラメータとして設定される値の型が記載される
メッセージの説明		メッセージの使い方や呼び出し時のモジュールの動作が記載される

4. 仕様書から状態遷移モデルの作成

この章ではインタフェース仕様書からブラックボックスなモジュールの状態遷移モデルを作成する方法を定義する。インタフェースのみが公開されるブラックボックスなモジュールを状態遷移モデルとしてモデル化するためには、モデルの状態と状態間の遷移が仕様書から抜き出す必要がある。前章のインタフェース仕様書の章の中で定義したようにモジュールがとる全状態は仕様書に記述され、かつそれが全てである.。よって状態遷移モデルの全状態はインタフェース仕様書に記述される状態を用いることができる。ブラックボックスなモジュールは外部に公開するメッセージを通じて操作され、それにより状態が遷移する。つまり、インタフェース仕様書のメッセージ仕様に記述されるモジュールの状態の変化を状態遷移としてモジュールのモデルに反映することで、ブラックボックスなモジュールの状態遷移モデルを作成することができる。状態間の遷移は遷移元状態、遷移先状態、トリガー、アクションの 4 つの要素から決まるので、この 4 つの要素の情報を仕様書から抜き出す必要がある。仕様書ではメッセージの形式により、Request 形式、Request-CallBack 形式、Event 形式の 3 つの形式があるため、それぞれのメッセージ形式について遷移元状態、遷移先状態、トリガー、アクションの 4 つの要素の情報抜き出す方法を定義する。

4.1. Request 形式の遷移

Request 形式のメッセージではモジュールに対して一方的にメッセージを送信する。このメッセージに対してモジュールが何らかの状態遷移がある場合、この遷移のトリガーはこのメッセージそのものである。よってトリガーにはこの Request 形式のメッセージの”メッセージ名”の情報が必要である。また、メッセージに設定可能なパラメータが存在する場合、設定されるパラメータによってトリガーとなるメッセージの意味も変わってくるため、トリガーの情報には”パラメータ”も必要である。Request 形式のメッセージがモジュールに対して送信されたとき、このメッセージに対しての応答メッセージはないため、呼び出し側のモジュールと呼ばれた側のモジュール間のメッセージに関して着目した場合、この遷移についてアクションはないことになる。よってアクションは”なし”となる。遷移元状態と遷移先状態はモジュールがどの状態で対象となる Request 形式のメッセージのを受け付けるか、メッセージを受けたときどの状態に遷移するかということであり、この情報はインタフェース仕様の”メッセージの説明”から抜き出す。以上をまとめると、Request 形式のメッセージ仕様において状態間の遷移を決めるための 4 つの情報と、抜き出し先となる仕様書の要素は表 4.1-1 のようになる。

表 4.1-1Request 形式の遷移を決めるための要素とメッセージ仕様項目の対応表

遷移を決めるための要素	抜き出し先のメッセージ仕様の項目
遷移元状態	メッセージの説明
遷移先状態	メッセージの説明

トリガー	メッセージ名、パラメータ
アクション	※なし

4.2. Request-CallBack 形式の遷移

Request-CallBack 形式のメッセージではモジュールに対してメッセージを送信し、それに対する応答のメッセージがある。このメッセージに対してモジュールが何らかの状態遷移がある場合、この遷移のトリガーはこのメッセージそのものである。よってトリガーにはこの Request-CallBack 形式のメッセージの”メッセージ名”から抜き出す。また、メッセージに設定可能なパラメータが存在する場合、設定されるパラメータによってトリガーとなるメッセージの意味も変わってくるため、トリガーの情報には”パラメータ”も必要である。Request-CallBack 形式のメッセージがモジュールに対して送信されたとき、このメッセージに対しての応答メッセージである CallBack メッセージが必ずモジュールから送信されるため、呼び出し側のモジュールと呼ばれた側のモジュール間のメッセージに関して着目した場合、この遷移についてのアクションは CallBack 形式のメッセージになる。よってアクションは CallBack 形式メッセージの”メッセージ名”から情報を抜き出す。また、アクションのメッセージにパラメータが設定される場合、その情報も必要となるため、”CallBack”形式メッセージの”パラメータ”の情報も必要となる。パラメータに設定される値は”メッセージの説明”から抜き出すため、”メッセージの説明”の情報も必要となる。遷移元状態と遷移先状態は Request 形式と同様の理由から情報はメッセージ仕様の”メッセージの説明”から抜き出す。以上をまとめると、Request-CallBack 形式のメッセージ仕様において状態間の遷移を決めるための 4 つの情報と、抜き出し先となる仕様書の要素は表 4.2-1 のようになる。

表 4.2-1 Request-CallBack 形式の遷移を決めるための要素とメッセージ仕様項目の対応表

遷移を決めるための要素	抜き出し先のメッセージ仕様の項目
遷移元状態	メッセージの説明
遷移先状態	メッセージの説明
トリガー	RequestCallBack 形式メッセージのメッセージ名 RequestCallBack 形式メッセージのパラメータ
アクション	CallBack 形式メッセージのメッセージ名 CallBack 形式メッセージのパラメータ メッセージの説明

4.3. Event 形式の遷移

Event 形式のメッセージでは仕様書で定義されるモジュール側から一方的に送信されるメッセージ形式である。呼び出し側のモジュールと呼ばれた側のモジュール間のメッセージに関して着目した場合、この遷移について呼び出し側のモジュールからは何のメッセージを送ることもなく、Event 形式のメッセージが来ることになるので、トリガーは”なし”となる。アクションはこの Event 形式のメッセージそのものであるので、Event 形式メッセージの”メッセージ名”と”パラメータ”から情報を抜き出す。パラメータに設定される値は”メッセージの説明”から抜き出すため、”メッセージの説明”の情報も必要となる。遷移元状態と遷移先状態は Request 形式と同様の理由から情報は”メッセージの説明”から抜き出す。以上をまとめると、Event 形式のメッセージ仕様において状態間の遷移を決めるための 4 つの情報と、抜き出し先となる仕様書の要素は表 4.3-1 のようになる。

表 4.3-1 Event 形式の遷移を決めるための要素とメッセージ仕様項目の対応表

遷移を決めるための要素	抜き出し先のメッセージ仕様の項目
遷移元状態	メッセージの説明
遷移先状態	メッセージの説明
トリガー	※なし
アクション	メッセージ名、パラメータ、メッセージの説明

5. メッセージ仕様の曖昧さについて

5.1. メッセージ仕様の曖昧さの整理

インタフェース仕様書の一般系について前項で定義した。このインタフェース仕様書の記述から状態遷移モデルを生成するために必要な要素である遷移元状態、遷移先状態、トリガー、アクションの 4 つの情報を読み取る。しかしながらインタフェース仕様書内のメッセージ仕様は自然言語の記述を含むため、必ずしも 4 つの情報が得られるわけではない。そこで本項ではメッセージタイプごとに分類して、遷移を決める 4 つの情報の中で曖昧さを含む可能性がある要素を整理する。

5.1.1. Request 形式

Request 形式は呼び出し側から対象モジュールに対する返却無しのメッセージを表している。そのためモジュール側から見たとき、トリガーは呼び出し側から来たメッセージが該当する。呼び出し側は設計者が設計した機能であり、このメッセージでどのようなパラメータが設定されるかはわかっているので、トリガーは「曖昧さが含まれる可能性がない部分」である。また、Request 形式は応答のメッセージはないため、アクションとしてのメッセージは常に「なし」となる。よってアクションは「曖昧さが含まれる可能性がない部分」であ

る。遷移元状態と遷移先状態は自然言語の説明部分に依存して決まるため、「曖昧さが含まれる可能性がある部分」に分けられる。以上の内容をまとめるとモデル生成のために必要な要素ごとの分類は次の表 5.1.1-1 のようになる。

表 5.1.1-1Request 形式の曖昧さが含まれる可能性の有無

遷移を決めるための要素	抜き出し先のメッセージ仕様の項目	曖昧さが含まれる可能性の有無(True/False)
遷移元状態	メッセージの説明	True
遷移先状態	メッセージの説明	True
トリガー	メッセージ名、パラメータ	False
アクション	※なし	False

5.1.2. Request – CallBack 形式

Request-CallBack 形式は呼び出し側からモジュールへのメッセージと、モジュールから呼び出し側への非同期の応答メッセージがある形式である。そのため Request 形式同様トリガーは「曖昧さが含まれる可能性がない部分」である。アクションはモジュール側から送られる CallBack のメッセージが該当するが、設定されるパラメータの値はモジュール側が設定するため、どのような値が設定されるかは自然言語である”メッセージの説明”の項目に依存する。よってアクションは「曖昧さが含まれる可能性がある部分」である。遷移元状態と遷移先状態は自然言語の説明部分に依存して決まるため、「曖昧さが含まれる可能性がある部分」に分けられる。以上の内容をまとめるとモデル生成のために必要な要素ごとの分類は次の表 5.1.2-1 のようになる。

表 5.1.2-1Request-CallBack 形式の曖昧さが含まれる可能性の有無

遷移を決めるための要素	抜き出し先のメッセージ仕様の項目	曖昧さが含まれる可能性の有無(True/False)
遷移元状態	メッセージの説明	True
遷移先状態	メッセージの説明	True
トリガー	RequestCallBack 形式メッセージのメッセージ名 RequestCallBack 形式メッセージのパラメータ	False
アクション	CallBack 形式メッセージのメッセージ名 CallBack 形式メッセージのパラメータ	True

	メッセージの説明	

5.1.3. Event 形式

Event 形式はモジュール側から呼び出し側へのメッセージである。そのためトリガーは常に「なし」であり、「曖昧さが含まれる可能性がない部分」である。アクションは Event 形式メッセージそのものが該当するが、設定されるパラメータの値はモジュール側が設定するため、どのような値が設定されるかは自然言語である”メッセージの説明”の項目に依存する。よってアクションは「曖昧さが含まれる可能性がある部分」である。遷移元状態と遷移先状態は自然言語の説明部分に依存して決まるため、「曖昧さが含まれる可能性がある部分」に分けられる。以上の内容をまとめるとモデル生成のために必要な要素ごとの分類は次の表 5.1.3-1 のようになる。

表 5.1.3-1Event 形式の曖昧さが含まれる可能性の有無

遷移を決めるための要素	抜き出し先のメッセージ仕様の項目	曖昧さが含まれる可能性の有無(True/False)
遷移元状態	メッセージの説明	True
遷移先状態	メッセージの説明	True
トリガー	※なし	False
アクション	メッセージ名、パラメータ、メッセージの説明	True

6. 曖昧さを含むメッセージ仕様の状態遷移モデルへの反映方法

本研究の手法では、メッセージ仕様から遷移を記述するために必要な要素の情報が得られない場合、その要素について取り得る全てのパターンの振る舞いを遷移としてモデルへ反映する。3 つの要素（遷移元状態、遷移先状態、アクション）がわからない可能性があるため、要素の組み合わせとしては、[遷移元状態]、[遷移先状態]、 [アクション]、[遷移元状態, 遷移先状態] 、 [遷移先状態, アクション]、[遷移元状態, アクション]、[遷移元状態, 遷移先状態, アクション]の 7 通りが存在する。それぞれの組み合わせのパターンにおける遷移をモデルへ反映させる方法を説明する。

6.1. 遷移元状態がわからない場合

メッセージ仕様の説明から遷移の遷移元状態がわからない場合、取り得る全ての状態からの遷移があるとしてモデルに反映させる。n 個の状態があり、そのうち遷移先の状態が k 個あるとする。このときどの遷移元状態から遷移先状態に対して遷移が存在するかがわからないため、本研究手法では n 個全ての状態から k 個の遷移先の状態に対してそれぞれ遷移が存在するとして、モデルに遷移を反映させる。遷移を反映させるイメージを図 6.1-1 に

示す。

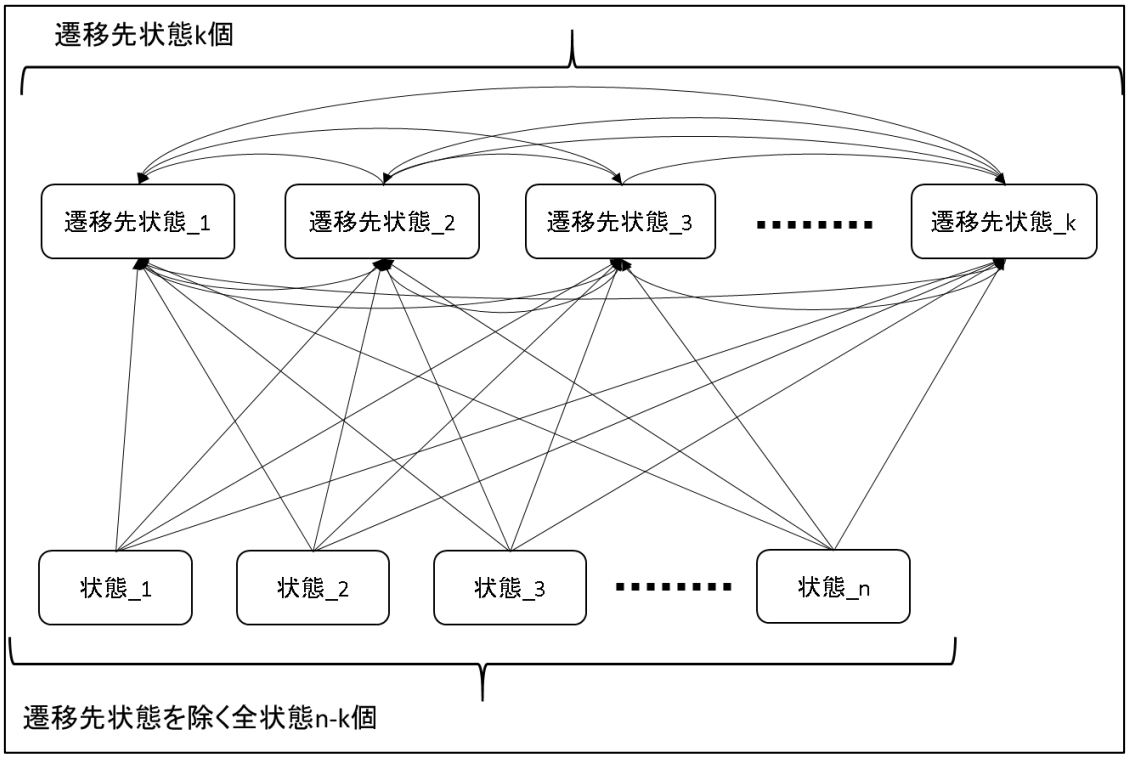

図 6.1-1 遷移元状態がわからないときの遷移をモデルに反映させるイメージ

具体的に遷移を決める情報のうちメッセージ仕様から抜き出した情報が以下の場合を考える。

遷移を決めるための要素	抜き出し結果
遷移元状態	[不明]
遷移先状態	STATE_B
トリガー	Trigger_X
アクション	Action_X

この例では遷移元状態のみわからず、遷移先・トリガー・アクションについてはメッセージ仕様から読み取れているとする。この遷移を状態遷移モデルに反映させようとするとき、遷移元はわからないため、元の状態遷移モデルで取り得る状態全て、すなわち STATE_A, STATE_B, STATE_C の 3 つ全てから遷移が起こりえるとして反映させる。反映後の状態遷移モデルは図 6.1-2 のようになる。

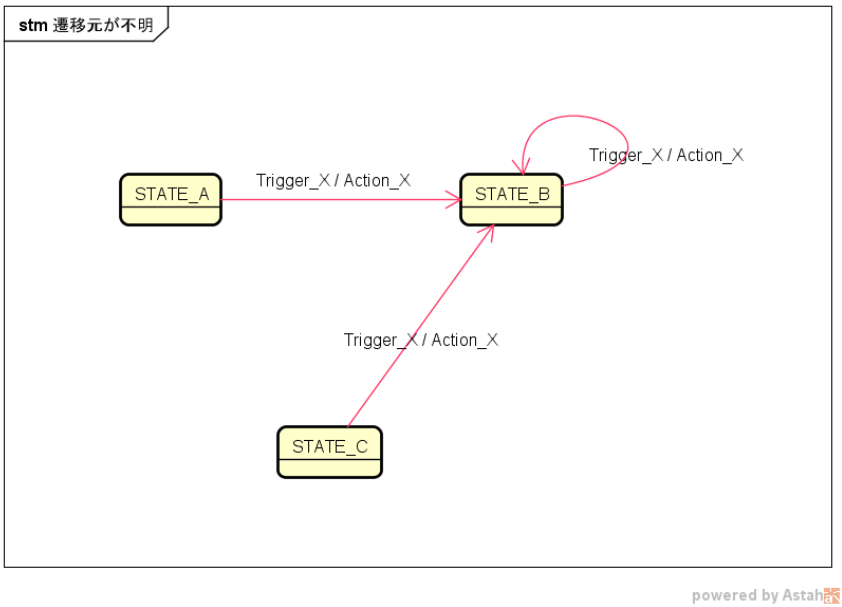

図 6.1-2 遷移元不明の遷移を反映

図の中の赤線が追加した遷移である。この例では遷移 STATE_A から STATE_B、STATE_B から STATE_B（自己遷移）、STATE_C から STATE_B の 3 つが追加されることになる。

6.2. 遷移先状態がわからない場合

メッセージ仕様の説明から遷移の遷移先の状態がわからない場合、取り得る全ての状態への遷移があるとしてモデルに反映させる。n 個の状態があり、そのうち遷移元の状態が k 個あるとする。このときどの遷移先状態に対して遷移が存在するかがわからないため、本研究手法では n 個全ての状態に対して k 個の遷移元態からそれぞれ遷移が存在するとして、モデルに遷移を反映させる。遷移を反映させるイメージを図 6.2-1 に示す。

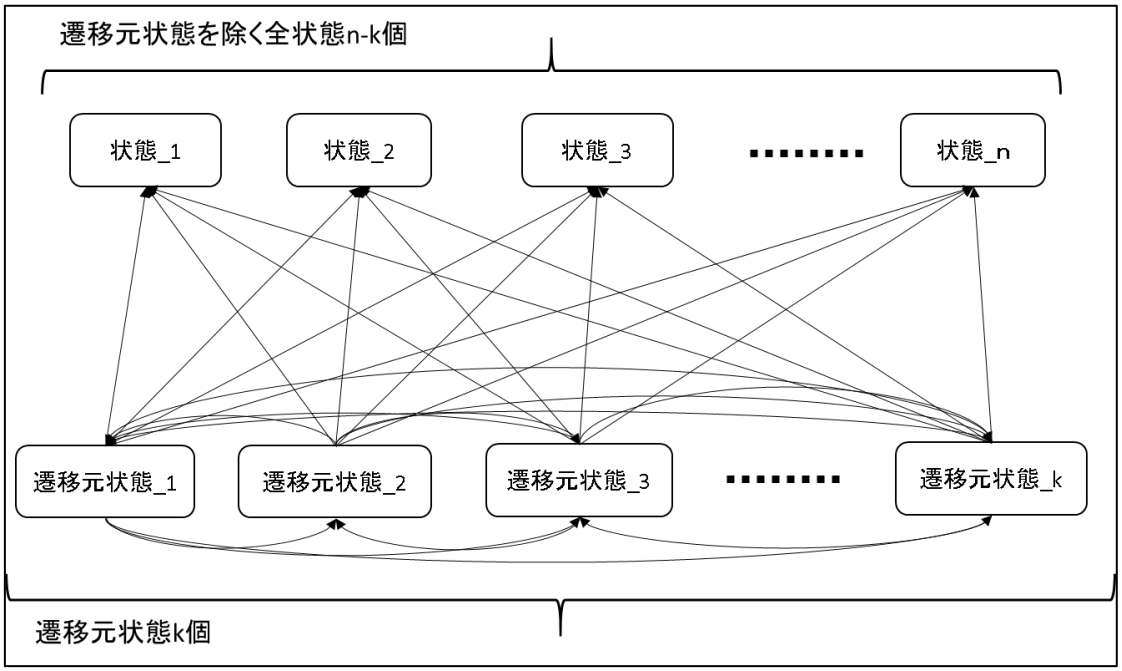

図 6.2-1 遷移先状態がわからないときの遷移をモデルに反映させるイメージ

具体的に遷移を決める情報のうちメッセージ仕様から抜き出した情報が以下の場合を考える。

遷移を決めるための要素	抜き出し結果
遷移元状態	STATE_C
遷移先状態	[不明]
トリガー	Trigger_X
アクション	Action_X

この例では遷移先状態のみわからず、遷移元状態・トリガー・アクションについてはメッセージ仕様から読み取れているとする。この振る舞いを状態遷移モデルに反映させようとするとき、遷移先はわからないため、元の遷移モデルで取り得る状態全て、すなわち STATE_A, STATE_B, STATE_C の 3 つの全てに対して遷移が起きえるとして反映させる。反映後の状態遷移モデルは図 6.2-2 のようになる。

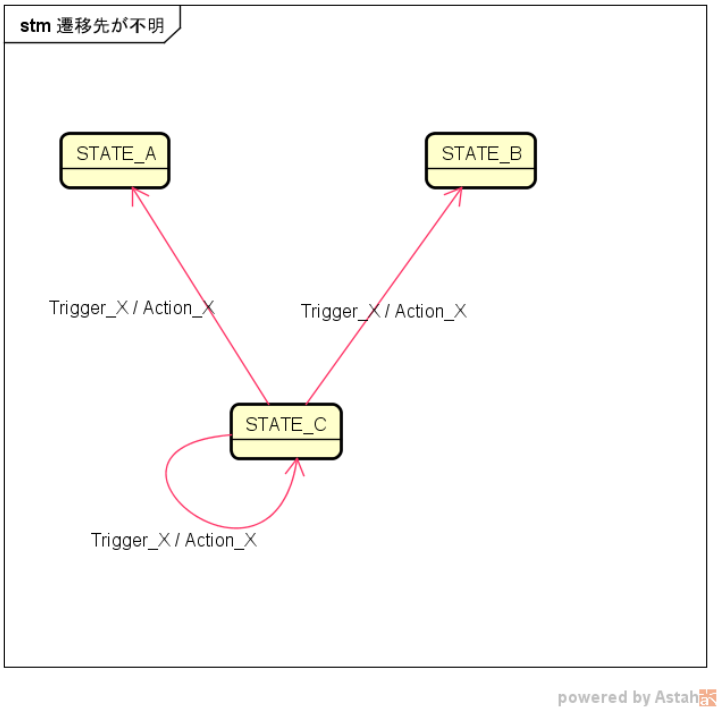

図 6.2-2 遷移先不明の遷移を反映

図の中の赤線が追加した遷移である。この例では遷移 STATE_C から STATE_A、STATE_C から STATE_B、STATE_C から STATE_C（自己遷移）の 3 つが追加されることになる。

6.3. アクションがわからない場合

メッセージ仕様の説明から振る舞いのアクションがわからない場合、取り得る全てのアクションが起こり得るとしてモデルに反映させる。遷移元状態と遷移先状態が決まっており、遷移のアクションに設定されうるパラメータの組み合わせが k 個あるがどのパラメータが設定されるかわからないとする。このときアクションとして取り得るパターンは k 種類あるため、本研究手法では k 個全てのアクションを遷移元状態と遷移先状態に反映させる。遷移を反映させるイメージを図 6.3-1 に示す。

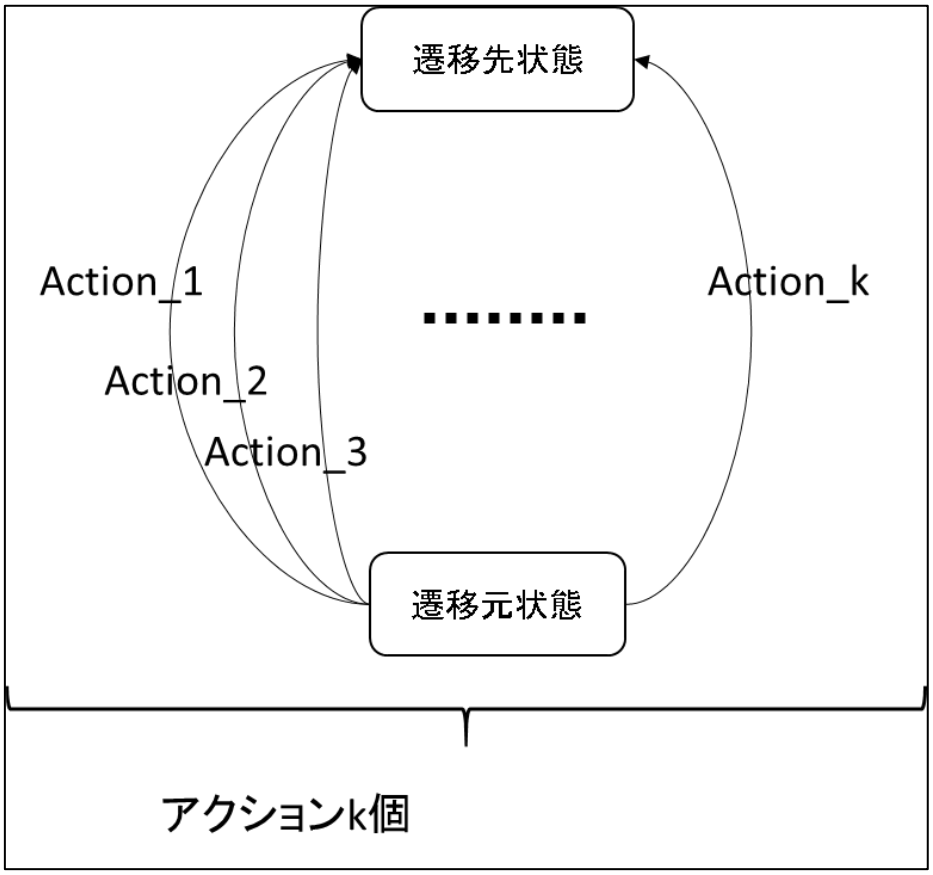

図 6.3-1 アクションがわからないときの遷移をモデルに反映させるイメージ

具体的に遷移を決める情報のうちメッセージ仕様から抜き出した情報が以下の場合を考える。

遷移を決めるための要素	抜き出し結果
遷移元状態	STATE_C
遷移先状態	STATE_A
トリガー	Trigger_X
アクション	[不明]

この例ではアクションのみわからず、遷移元状態・遷移先状態・トリガーについてはメッセージ仕様から読み取れているとする。この振る舞いを状態遷移モデルに反映させようとするとき、アクションはわからないため、起こり得るアクション全てで遷移が起こり得るとして反映させる。ここで全項の仕様書の分析から、アクションのわからなさはとり得るパラメータの範囲内でのわからなさであることがわかっているので、取り得る範囲はアクションとなるメッセージのパラメータの範囲となる。アクションの範囲を Action_X, Action_Y,

Action_Z とすると、反映後の状態遷移モデルは図 6.3-2 のようになる。

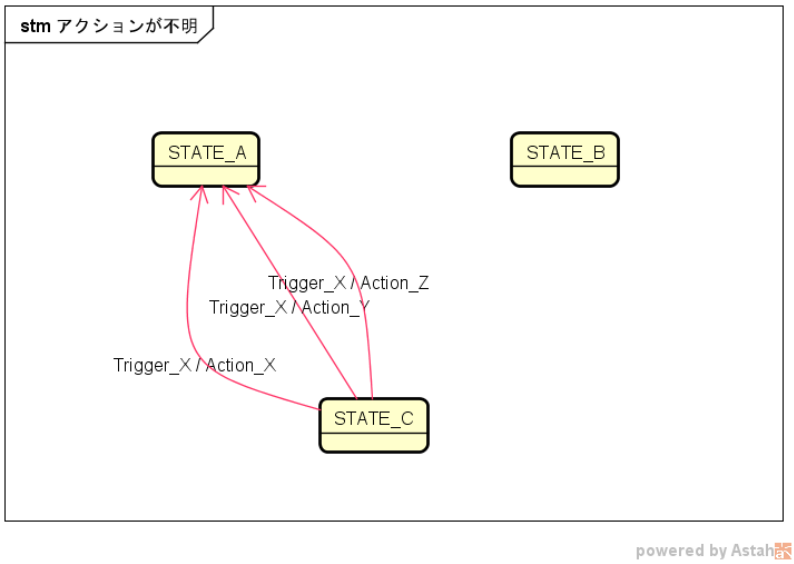

図 6.3-2 アクション不明の遷移を反映

図の中の赤線が追加した遷移である。この例では状態 STATE_C から STATE_A でアクションがそれぞれ異なる遷移 3 つが追加されることになる。

6.4. 遷移元状態，遷移先状態がわからない場合

メッセージ仕様の説明から遷移元状態、遷移先状態がわからない場合、取り得る全ての状態を遷移元状態・遷移先状態として掛け合わせた遷移をモデルに反映させる。n 個の状態がある場合、本研究手法では n 個全ての状態を遷移元状態及び遷移先状態としてモデルに遷移を反映させる。遷移を反映させるイメージを図 6.4-1 に示す。

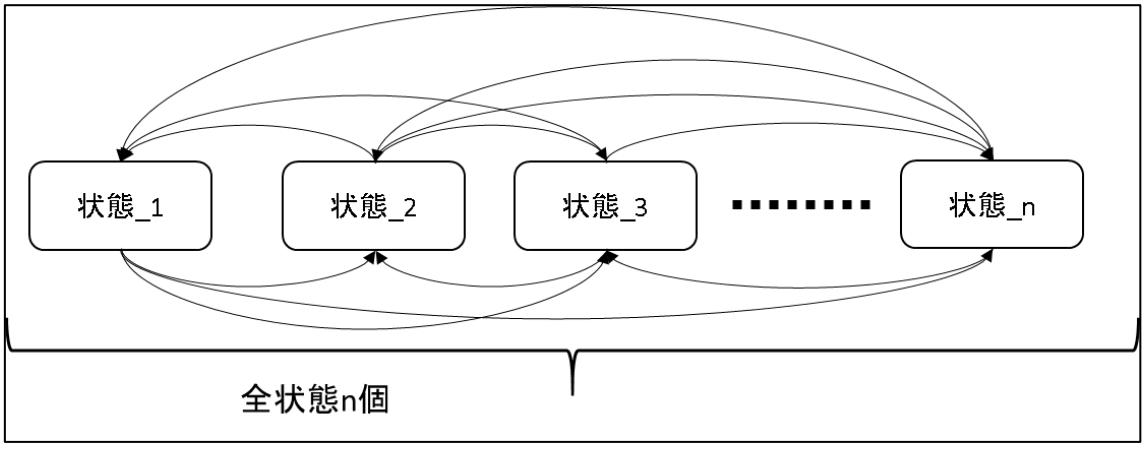

図 6.4-1 遷移元・遷移先状態がわからないときの遷移をモデルに反映させるイメージ

具体的に遷移を決める情報のうちメッセージ仕様から抜き出した情報が以下の場合を考

える。

遷移を決めるための要素	抜き出し結果
遷移元状態	[不明]
遷移先状態	[不明]
トリガー	Trigger_X
アクション	Action_X

この例では遷移元状態・遷移先状態がメッセージ仕様から読み取れず、トリガーとアクションがメッセージ仕様から読み取れているとする。この振る舞いを状態遷移モデルに反映させようとするとき、取り得る全ての状態を遷移元・遷移先状態として組み合わせた遷移を考慮する必要がある。この例の場合、遷移の組み合わせとしては取り得る状態 STATE_A、STATE_B、 STATE_C の内積の数だけ存在することになる。反映後の状態遷移モデルは図 6.4-2 のようになる。

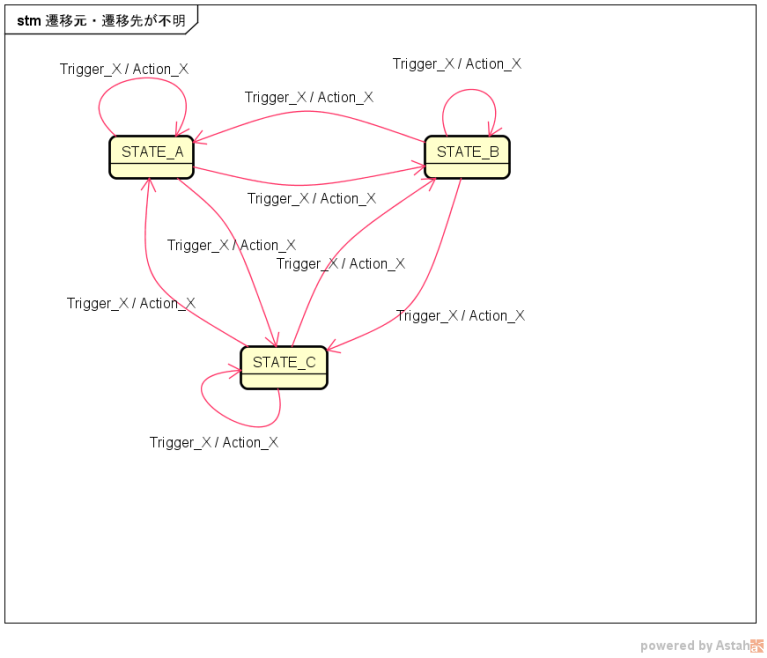

図 6.4-2 遷移先、遷移元不明の遷移を反映

図の中の赤線が追加した遷移である。遷移の組み合わせを{遷移元状態, 遷移先状態}として表すとき、組み合わせは{STATE_A, STATE_A}, {STATE_A, STATE_B}, {STATE_A, STATE_C}, {STATE_B, STATE_A}, {STATE_B, STATE_B}, {STATE_B, STATE_C}, {STATE_C, STATE_A}, {STATE_C, STATE_B}, {STATE_C, STATE_C}の 9 つが追加されることになる。

6.5. 遷移先状態、アクションがわからない場合

メッセージ仕様の説明から遷移先状態、アクションがわからない場合、まず取り得る全ての状態が遷移先状態としてモデルに反映させる。遷移元状態は決まっているため、遷移元状態と遷移先状態の組み合わせは、全状態の数分だけ存在することになる。またアクションが不明であるから、メッセージのパラメータの範囲だけアクションがとり得る範囲も存在する。そのため遷移元状態と遷移先状態の組み合わせに加え、アクションの取り得る範囲も加えて遷移を反映させる。以下にメッセージ受信時の遷移をモデルに反映させる方法を示す。

具体的に遷移を決める情報のうちメッセージ仕様から抜き出した情報が以下の場合を考える。

遷移を決めるための要素	抜き出し結果
遷移元状態	STATE_A
遷移先状態	[不明]
トリガー	Trigger_X
アクション	[不明]

この例では遷移先状態とアクションがメッセージ仕様から読み取れず、遷移元状態とトリガーがメッセージ仕様から読み取れているとする。この例の場合、遷移の組み合わせを{遷移元状態, 遷移先状態}として表すとき、組み合わせは{STATE_A, STATE_A}, {STATE_A, STATE_B}, {STATE_A, STATE_C}の 3 つがある。またアクションの範囲を Action_X, Action_Y, Action_Z とすると、前述の{遷移元状態, 遷移先状態}の組み合わせごとに 3 つのアクションの遷移が存在するとしてモデルに反映させる。反映後の状態遷移モデルは図 6.5-1 のようになる。

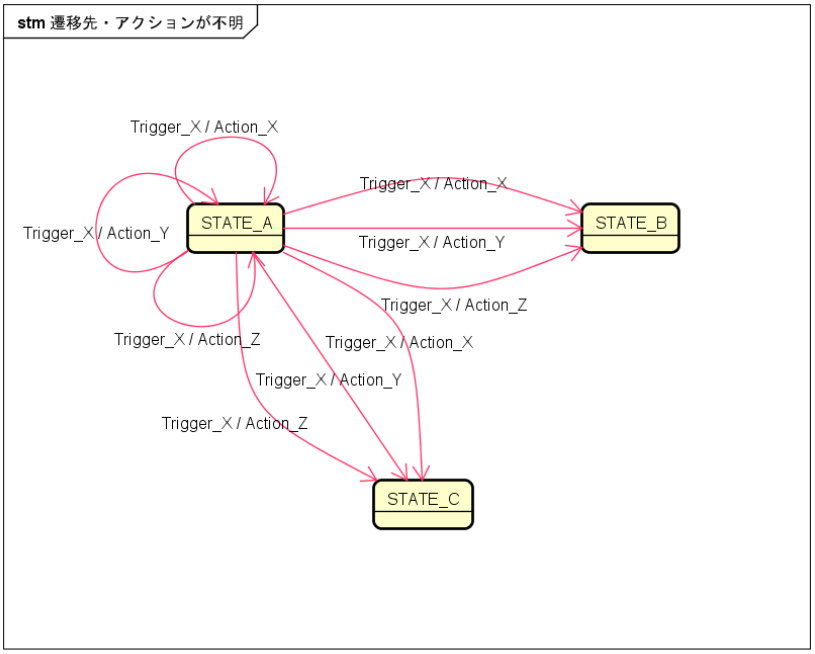

図 6.5-1 遷移先、アクション不明の遷移を反映

6.6. 遷移元状態、アクションがわからない場合

メッセージ仕様の説明から遷移元状態、アクションがわからない場合、まず取り得る全ての状態が遷移元状態としてモデルに反映させる。遷移先状態は決まっているため、遷移元状態と遷移先状態の組み合わせは、全状態の数分だけ存在することになる。またアクションが不明であるから、メッセージのパラメータの範囲だけアクションがとり得る範囲も存在する。そのため遷移元状態と遷移先状態の組み合わせに加え、アクションの取り得る範囲も加えて遷移を反映させる。以下にメッセージ受信時の遷移をモデルに反映させる方法を示す。

具体的に遷移を決める情報のうちメッセージ仕様から抜き出した情報が以下の場合を考える。

遷移を決めるための要素	抜き出し結果
遷移元状態	[不明]
遷移先状態	STATE_C
トリガー	Trigger_X
アクション	[不明]

この例の場合、遷移の組み合わせを{遷移元状態, 遷移先状態}として表すとき、組み合わせは{STATE_A, STATE_C}, {STATE_B, STATE_C}, {STATE_C, STATE_C}の 3 つがある。またアクションの範囲を Action_X, Action_Y, Action_Z とすると、前述の{遷移元状態, 遷移先状態}の組み合わせごとに 3 つのアクションの遷移が存在するとしてモデルに反映させ

る。反映後の状態遷移モデルは図 6.6-1 のようになる。

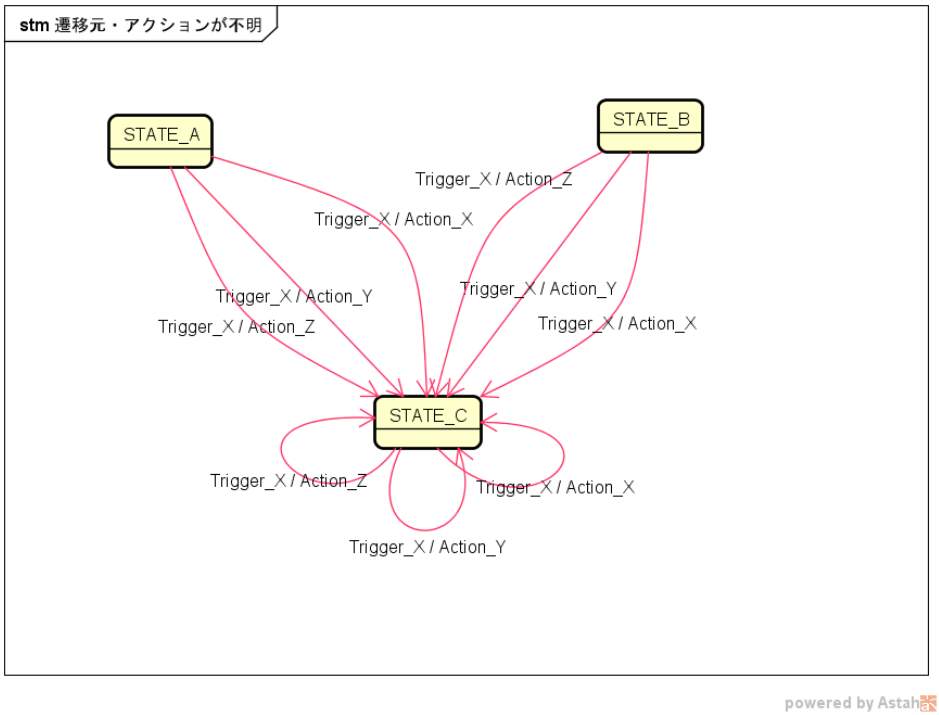

図 6.6-1 遷移元、アクション不明の遷移を反映

6.7. 遷移先状態、遷移元状態、アクションがわからない場合

メッセージ仕様の説明から遷移元状態、遷移元状態、アクションがわからない場合、まず取り得る全ての状態が遷移先状態、遷移元状態になり得るとしてモデルに反映させる。遷移元状態と遷移先状態の組み合わせは、状態の内積の数分だけ存在することになる。またアクションが不明であるから、メッセージのパラメータの範囲だけアクションがとり得る範囲も存在する。そのため遷移元状態と遷移先状態の組み合わせに加え、アクションの取り得る範囲も加えて遷移を反映させる。以下にメッセージ受信時の遷移をモデルに反映させる方法を示す。

具体的に遷移を決める情報のうちメッセージ仕様から抜き出した情報が以下の場合を考える。

遷移を決めるための要素	抜き出し結果
遷移元状態	[不明]
遷移先状態	[不明]
トリガー	Trigger_X
アクション	[不明]

この例の場合、遷移の組み合わせを{遷移元状態，遷移先状態}として表すとき、組み合わせ

は{STATE_A, STATE_A}, {STATE_A, STATE_B}, {STATE_A, STATE_C}, {STATE_B, STATE_A}, {STATE_B, STATE_B}, {STATE_B, STATE_C}, {STATE_C, STATE_A}, {STATE_C, STATE_B}, {STATE_C, STATE_C}の 9 つがある。またアクションの範囲を Action_X, Action_Y, Action_Z とすると、前述の{遷移元状態, 遷移先状態}の組み合わせごとに 3 つのアクションの遷移が存在するとしてモデルに反映させる。反映後の状態遷移モデルは図 6.7-1 のようになる。

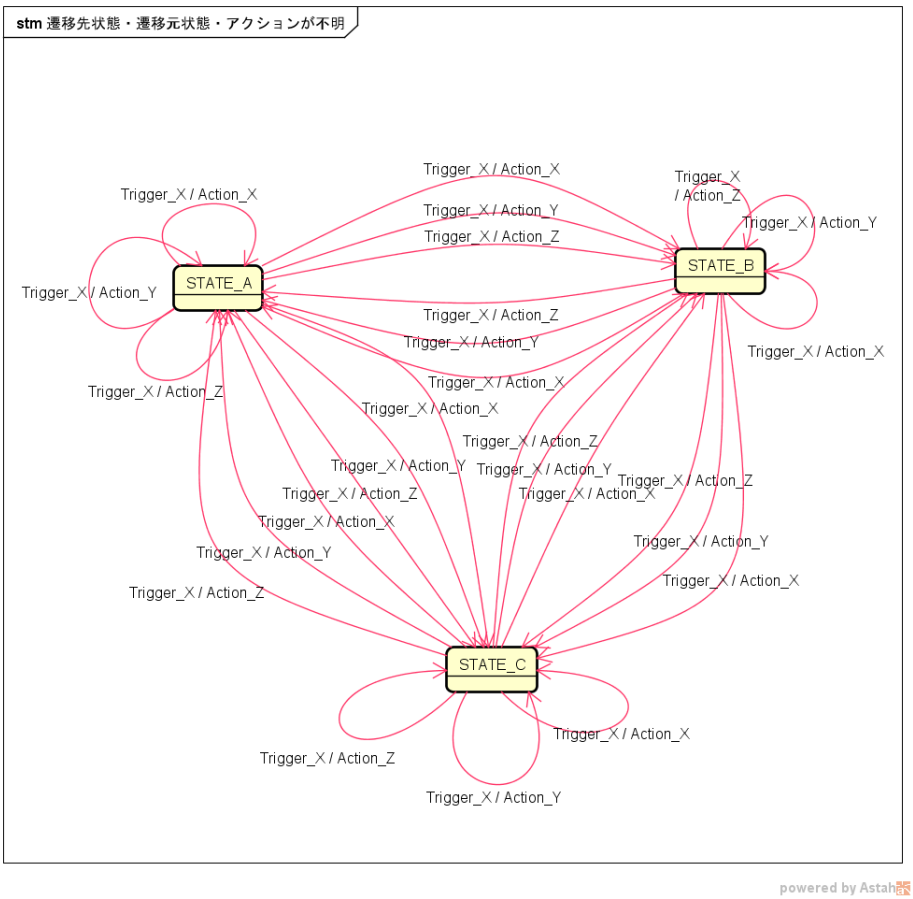

図 6.7-1 遷移元、遷移先、アクション不明の遷移を反映

7. 実験

7.1. 実験の概要

本研究手法の有効性を示すための適応実験を行った。実験では実際の開発であった車載オーディオの組み込みシステム開発を抽象化したものを題材にして行った。題材となるシステム開発の設計には実際の開発であったものと同じ不具合があらかじめ含まれている。実験では本研究の提案手法である仕様書からのモデル生成と合わせてシステム全体を状態遷移モデル化し、さらにそのモデルに対してモデル検査を実施した。モデル検査によって潜在する不具合を検知することができ、本手法の有効性を示すことができた。

7.2. 実験題材

7.2.1. 実験題材の概要

実験題材となるシステムは車載オーディオシステム（以下、オーディオシステムと呼ぶ）である。オーディオシステムは自動車内でユーザーの操作に応じて音楽の再生、停止、早送りなどの制御を行うシステムである。オーディオシステムは、UI 層と App 層と MW 層の 3 つのモジュールで構成され、それぞれのモジュールはメッセージを通じて並行協調動作する。UI 層はユーザーの操作に応じて対応するメッセージを App 層に対して通知する。App 層は UI 層からのメッセージをもとに MW 層で定義される API を呼び出す。また、MW 層から送信されるメッセージを受けて App 層の状態を制御する。MW 層はハードウェアを操作するためのモジュールであり、MW 層を通じて音楽再生を行う。図 7.2.1-1 に本システムのソフトウェア構成を示す。

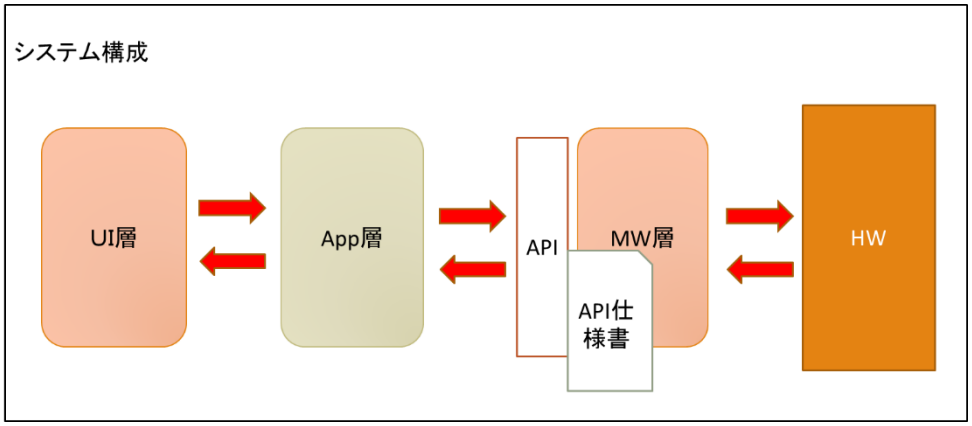

図 7.2.1-1 システム構成

システムは大きく UI 層、App 層、MW 層に分けられている。各層はそれぞれ非同期で並行動作しており、各層の間はメッセージを介してやり取りされる。UI 層は App 層と、App 層は UI 層・MW 層と、MW 層は App 層とメッセージのやり取りを行う。UI 層・App 層は開発において設計者側が内部の動作を把握しており正確にモデル化可能なモジュールであるが、MW 層は外部から提供されたモジュールであるため、内部の詳細な動作を開発者は把握することができない。そのため、インタフェース仕様書という形でインタフェースが公開されており、これを呼び出すことで App 層から MW 層を操作し、システムの機能を実現している。各層の詳細について説明する。

7.2.2. UI 層

UI 層はユーザーの操作を伝える部分である。ユーザーがオーディオ機器を操作したときの要求（再生、停止など）を App 層に対してメッセージとして送信する。また機器を操作する際のユーザーの操作の制約もこの層で制御する。例えば「再生ボタンを押した後は再度

再生ボタンを連続で押すことはできない」等が操作の制約に当てはまる。本オーディオシステムで実現させる機能について、機器で可能な操作とそれによって実現させる機能の対応表を表 7.2.2-1 に示す。

表 7.2.2-1 オーディオシステムで可能な操作と対応する機能

操作名	機能
再生開始	通常速度再生を開始する
再生停止	通常速度再生を停止する
早送り開始	通常速度再生を早送り再生する
早送り解除	早送り再生を解除し、通常速度再生に戻す

また、機器を操作する際の制約は表 7.2.2-2 のようになる。

表 7.2.2-2 ユーザー操作の制約

操作の制約
再生停止中のみ「再生開始」の操作を実現できる
通常速度再生中のみ「再生停止」の操作を実行できる
通常速度再生中のみ「早送り開始」の操作を実行できる
早送り再生中のみ「早送り解除」の操作を実行できる

7.2.3. UI 層と App 層の通信

UI 層と App 層は非同期のメッセージで通信する。メッセージは UI 層で受けたユーザーの操作と 1 対 1 で対応し、ユーザー操作が起きるとそれに対応したメッセージが UI 層から App 層へ送信される。UI-App 間における操作に対応するメッセージは表 7.2.3-1 のように定義される。

表 7.2.3-1UI-App 間の操作ごとのメッセージ

操作名	対応メッセージ
再生開始	EVENT_PLAY
再生停止	EVENT_STOP
早送り開始	EVENT_FF
早送り解約	EVENT_FF_CANCEL

7.2.4. App 層

App 層は UI 層と MW 層の間にあり、UI 層から来た要求メッセージに応じて適切な API メッセージを MW 層に対して送信する。App 層は開発での設計対象のモジュールである。オーディオシステムの機能を実現させるためにモジュールの協調動作を制御できるよう App 層を設計する。設計時には協調動作時のメッセージのやり取りをシーケンス図で表す。例えば停止中のオーディオ機器を再生する場合の協調動作を表したシーケンス図は図 7.2.4-1 のようになる。

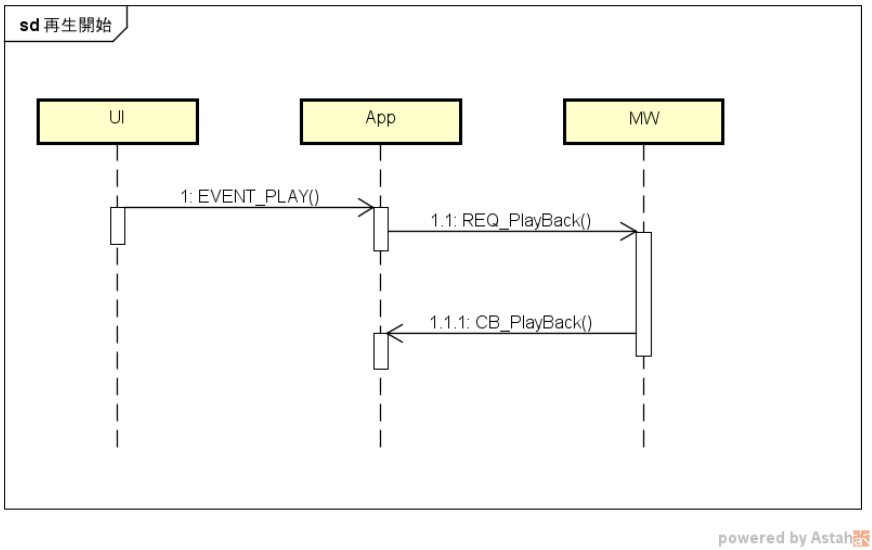

図 7.2.4-1 再生開始シーケンス

再生開始時には先ず UI 層から EVENT_PLAY のメッセージが App 層に対して送信される。App 層はこのメッセージを送信すると、再生開始の MW 層のインタフェースである REQ_PlayBack メッセージを MW 層に送る。このとき App 層は初期状態の STANBY 状態である。このメッセージ送信後 MW 層は REQ_PlayBack メッセージのコールバックである CB_PlayBack メッセージを App 層に対して送信する。この間 App 層は CB_PlayBack メッセージを待つための WAIT_PLAY 状態に遷移する。CB_PlayBack を App 層が受けると再生開始処理が完了したため、App 層は PLAY 状態に遷移する。このようにシステムの機能を実現するために 3 層がメッセージの送受信を通じて協調動作し、このメッセージの一連のやり取りをシーケンス図で表す。App 層ではどの状態でどのメッセージが来た時にどのように動作するかを設計時に考える。App 層における状態一覧は表 7.2.4-1 のようになる。また、この App 層の制御を表現したものが状態遷移表である。状態遷移表には App の状態を縦の列に、App が受ける UI、MW からのメッセージを横の行に書き並べる。それぞれの状態ごとで受けたメッセージごとに実行する動作と遷移先の状態を各遷移表のマスに記述する。実験題材の開発での状態遷移表を表 7.2.4-2 に示す。

表 7.2.4-1App 層で定義される状態一覧

状態一覧
WAIT_STOP
STANBY
WAIT_PLAY
PLAY
WAIT_FAST_FORWARD
FAST_FORWARD
WAIT_FF_CANCEL

表 7.2.4-2 状態遷移表

受信メッセージ／App層の状態	WAIT_STOP	STANBY	WAIT_PLAY	PLAY	WAIT_FAST_FORWARD	FAST_FORWARD	WAIT_FF_CANCEL
EVENT_PLAY	N/A	REQ_PlayBack呼び出し WAIT_PLAY状態へ遷移	N/A	N/A	N/A	N/A	N/A
EVENT_FF	N/A	N/A	N/A	REQ_FastForward呼び出し WAIT_FAST_FORWARD 状態へ遷移	N/A	N/A	N/A
EVENT_FF_CANCEL	N/A	N/A	N/A	N/A	N/A	REQ_FastForwardCancel 呼び出し WAIT_FF_CANCEL 状態へ遷移	N/A
EVENT_STOP	N/A	N/A	N/A	REQ_PlayStop呼び出し WAIT_STOP状態へ遷移	N/A	N/A	N/A
CB_PlayBack	N/A	N/A	PLAY状態へ遷移	N/A	N/A	N/A	N/A
CB_FastForward	N/A	N/A	N/A	N/A	FAST_FORWARD状態へ遷移	N/A	N/A
CB_FastForward Cancel	N/A	N/A	N/A	N/A	N/A	N/A	PLAY状態へ遷移
CB_Stop	STANBY 状態へ遷移	N/A	N/A	N/A	N/A	N/A	N/A
Notify_Status	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移

図 7.2.4-1 のシーケンスを実現するために App 層は STANBY 状態で EVENT_PLAY を受けたとき、REQ_PlayBack を呼び出し、WAIT_PLAY 状態へ遷移する必要がある。表 7.2.4-2 を見ると、表の STANBY の列で行が EVENT_PLAY のマスに「REQ_PlayBack 呼び出し」と「WAIT_PLAY 状態へ遷移」の記述がある。このように App がとり得る状態と受けるメッセージごとの動作を全て遷移表に記述することで、App の動作に漏れがないようにしている。

7.2.5. MW 層

MW 層は外部から提供されたモジュールであり、内部の設計や実装は公開されておらず、詳細な振る舞いを App 層の設計者は知ることはできない。MW 層はインタフェース仕様書という形でインタフェースが公開されており、この仕様書をもとに App 層の設計者は MW 層の振る舞いを把握する。MW 層のモジュールは音楽再生に関する機能を呼び出すための音楽再生や再生停止などのインタフェースが提供されている。App 層の設計者はユーザーからの操作をもとにこの MW 層のインタフェースのメッセージを送信する。音楽再生などのメッセージは Request-CallBack 形式になっており、実際に MW 層のモジュールにメッセージを送信した後、完全に動作が開始するタイミングを CallBack メッセージで知ることができる。このモジュールには Request-CallBack 形式で定義された再生開始、再生停止、早送り開始、早送り解除のメッセージと、Event 形式のモジュール状態通知メッセージがある。メッセージ送信することにより MW 層のモジュールの状態は遷移する。状態は再生状態、非再生状態、早送り状態の 3 つが仕様書に記載されている。MW 層がとり得る状態を表 7.2.5-1 に示す。

表 7.2.5-1MW 層がとり得る全状態

MWState
再生状態
非再生状態
早送り状態

仕様書にはインタフェースとして 4 つの Request-CallBack 形式のメッセージが 4 つ Event 形式のメッセージが 1 つ記述されている。以下にメッセージを記述する。

表 7.2.5-2 再生開始インタフェース

パラメータ		記述される内容
メッセージ形式		Request-CallBack(Request)
メッセージ名		REQ_PlayBack
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		非再生状態で本 API を呼び出すと再生開始を要求する。再生開始完了後、再生状態になると CB_PlayBack メッセージを返す。

パラメータ	記述される内容

メッセージ形式		Request-CallBack(CallBack)
メッセージ名		CB_PlayBack
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		再生開始完了時に本 API を返す。

表 7.2.5-3 早送り開始インタフェース

パラメータ		記述される内容
メッセージ形式		Request-CallBack(Request)
メッセージ名		REQ_FastForward
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		再生状態で本APIを呼び出すと早送り開始を要求する。早送り開始完了後、早送り状態になるとCB_FastForward メッセージを返す。

パラメータ		記述される内容
メッセージ形式		Request-CallBack(CallBack)
メッセージ名		CB_FastForward
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		早送り開始完了時に本 API を返す。

表 7.2.5-4 再生停止インタフェース

パラメータ		記述される内容
メッセージ形式		Request-CallBack(Request)
メッセージ名		REQ_PlayStop
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		再生状態で本 API を呼び出すと再生停止を要求する。再生停止完了後、非再生状態になると CB_PlayStop メッセージを返す。

パラメータ	記述される内容
メッセージ形式	Request-CallBack(CallBack)

メッセージ名		CB_PlayStop
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		再生停止完了時に本 API を返す。

表 7.2.5-5 早送り解除インタフェース

パラメータ		記述される内容
メッセージ形式		Request-CallBack(Request)
メッセージ名		REQ_FastForwardCancel
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		早送り状態で本 API を呼び出すと早送り解除を要求する。 早送り解除完了後、再生状態になると CB_メッセージを返す。

パラメータ		記述される内容
メッセージ形式		Request-CallBack(CallBack)
メッセージ名		CB_FastForwardCancel
パラメータ	パラメータ数	0
	パラメータの型	None
メッセージの説明		早送り解除完了時に本 API を返す。

表 7.2.5-6 状態通知インタフェース

パラメータ		記述される内容
メッセージ形式		Event
メッセージ名		Notify_Status
パラメータ	パラメータ数	1
	パラメータの型	MWState:{Stop, Play, FastForward}
メッセージの説明		MW の現在の状態を通知する。

7.2.6. 実験題材に潜在する不具合

題材となるシステム開発の設計には実際に筆者が経験したものと同じ不具合を含め、いくつかの不具合が含まれている。この項では不具合の内容について説明する。不具合は設計上の考慮漏れにより発生した。図 7.2.6-1 は再生状態から早送りを開始し、早送りを解除す

るまでのシーケンス図である。

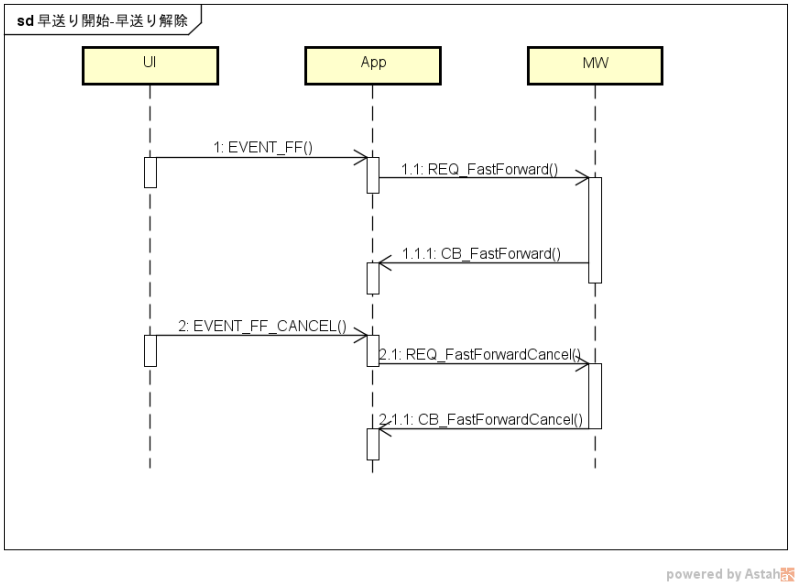

図 7.2.6-1 早送り開始-解除時のシーケンス

このシーケンス図のメッセージの際の App 層の状態遷移と処理の流れは表 7.2.6-1 のようになる。先ず UI 層から早送り開始を通知する EVENT_FF メッセージが App 層に送られる。このとき App 層は PLAY 状態であるので、REQ_FastForward を呼び出し、WAIT_FAST_FORWARD 状態へ遷移する。その後 CB_FastForward メッセージが MW 層から App 層へ送信されると App 層は FAST_FORWARD 状態に遷移する。早送り解除も同様に UI 層から App 層へ EVENT_FF_CANCEL メッセージが通知される。App 層は REQ_FastForwardCancel を呼び出し、WAIT_FF_CANCEL 状態へ遷移する。その後 CB_FastForwardCancel メッセージが MW 層から App 層へ送信されると App 層は PLAY 状態へ遷移し、早送り解除が完了する。

表 7.2.6-1 早送り開始-解除時の状態遷移表の流れ

受信メッセージ／App層の状態	WAIT_STOP	STANBY	WAIT_PLAY	PLAY	WAIT_FAST_FORWARD	FAST_FORWARD	WAIT_FF_CANCEL
EVENT_PLAY	N/A	REQ_PlayBack呼び出し WAIT_PLAY状態へ遷移	N/A	N/A	N/A	N/A	N/A
EVENT_FF	N/A	N/A	N/A	REQ_FastForward呼び出し WAIT_FAST_FORWARD 状態へ遷移	N/A	N/A	N/A
EVENT_FF_CANCEL	N/A	N/A	N/A	N/A	N/A	REQ_FastForwardCancel 呼び出し WAIT_FF_CANCEL 状態へ遷移	N/A
EVENT_STOP	N/A	N/A	N/A	REQ_PlayStop呼び出し WAIT_STOP状態へ遷移	N/A	N/A	N/A
CB_PlayBack	N/A	N/A	PLAY状態へ遷移	N/A	N/A	N/A	N/A
CB_FastForward	N/A	N/A	N/A	N/A	FAST_FORWARD状態へ遷移	N/A	N/A
CB_FastForward Cancel	N/A	N/A	N/A	N/A	N/A	N/A	PLAY状態へ遷移
CB_Stop	STANBY状態へ遷移	N/A	N/A	N/A	N/A	N/A	N/A
Notify_Status	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移

次に不具合が起きたときのシーケンスを図 7.2.6-2 に示す。このシーケンスでは PLAY 状態の App 層に対して UI 層から早送り開始を通知する EVENT_FF メッセージが送られる。その直後に MW 層から Notify_Status と呼ばれる MW 層の状態を通知するメッセージが App 層へ送られる。App 層は通常の早送り開始と同様に REQ_FastForward を呼び出し、WAIT_FAST_FORWARD 状態へ遷移する。CB_FastForward メッセージが MW 層から App 層へ送信される。その後早送り解除を知らせる EVENT_FF_CANCEL メッセージが UI 層から App 層へ通知されるが、App 層からは何のメッセージも送信されず、早送り解除の処理が始まらない。この時の App 層の状態遷移と処理の流れを表 7.2.6-2 に示す。

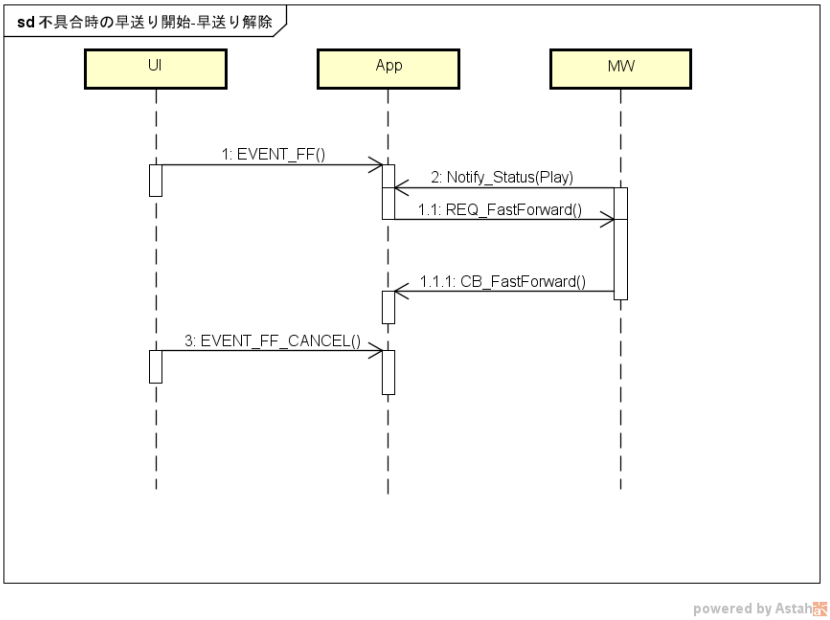

図 7.2.6-2 不具合時の早送り開始-解除シーケンス

表 7.2.6-2 不具合時の早送り開始-解除時の状態遷移表の流れ

受信メッセージ／App層の状態	WAIT_STOP	STANBY	WAIT_PLAY	PLAY	WAIT_FAST_FORWARD	FAST_FORWARD	WAIT_FF_CANCEL
EVENT_PLAY	N/A	REQ_PlayBack呼び出し WAIT_PLAY状態へ遷移	N/A	N/A	N/A	N/A	N/A
EVENT_FF	N/A	N/A	N/A	REQ_FastForward呼び出し WAIT_FAST_FORWARD 状態へ遷移	N/A	N/A	N/A
EVENT_FF_CANCEL	N/A	N/A	N/A	N/A	N/A	REQ_FastForwardCancel 呼び出し WAIT_FF_CANCEL 状態へ遷移	N/A
EVENT_STOP	N/A	N/A	N/A	REQ_PlayStop呼び出し WAIT_STOP状態へ遷移	N/A	N/A	N/A
CB_PlayBack	N/A	N/A	PLAY状態へ遷移	N/A	N/A	N/A	N/A
CB_FastForward	N/A	N/A	N/A	N/A	FAST_FORWARD状態へ遷移	N/A	N/A
CB_FastForward Cancel	N/A	N/A	N/A	N/A	N/A	N/A	PLAY状態へ遷移
CB_Stop	STANBY 状態へ遷移	N/A	N/A	N/A	N/A	N/A	N/A
Notify_Status	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移	通知された状態へ遷移

何故早送り解除の処理が始まらないかというと、先ず早送り開始時には App 層は REQ_FastForward を呼び出し、WAIT_FAST_FORWARD 状態へ遷移する。その直後

Notify_Status の状態通知メッセージが App 層に送信されるが、このとき UI 層は早送りを開始していない通常の再生中であるため、パラメータに Play が設定される。このメッセージを App 層が受けると、設計上「通知された状態へ遷移」となっており、App 層も PLAY 状態へ遷移する。その後 CB_FastForward メッセージが MW 層から App 層へ送信されるが、PLAY 状態で CB_FastForward メッセージを受けた際の処理は何もないため、App 層はそのまま PLAY の状態にとどまる。さらにその後 EVENT_FF_CANCEL メッセージを App 層が受けるが、PLAY 状態で EVENT_FF_CANCEL メッセージを受けた際の処理もないため、結果的に早送り解除の処理を無視することになり、不具合につながった。設計時には把握できていなかったが、Notify_Status はオーディオ再生中に再生楽曲の曲と曲がまたいだタイミングで通知されることがわかった。例えば再生中に曲を 2 回またいだ場合は Notify_Status(Play)のメッセージが UI 層から 2 回通知される。早送り中に 1 回曲を跨いだ場合は、Notify_Status(FastForward)のメッセージが 1 回通知される。このメッセージは単純に再生中や早送り中に来ただけでは App 層は同じ状態にとどまるだけで不具合になることはない。しかし、図 7.2.6-2 のような操作時のイベントとタイミングが重なった場合などに不具合につながり、設計時に不具合を見つけることが困難な不具合である。

7.3. 検査対象のモデル化

モデル検査を適用するためにシステムを状態遷移モデルとして記述する。システムは 3 層で構成されており、それぞれを状態遷移モデルで記述する。モデル全体としてはフォーマルメソッド導入ガイダンス[3]のコンポーネント駆動をもとに、システムの構成要素と要素間の関係に着目し、UI 層、MW 層、App 層をそれぞれモデル化した。

7.3.1. MW 層のモデル化

MW 層は仕様書から状態遷移モデルを生成する。全状態は仕様書に定義されておりこの仕様書では再生状態、非再生状態、早送り状態の 3 つが該当する。次に状態間の遷移は仕様書の API から判断する必要があり、この仕様書では再生開始インタフェース、再生停止インタフェース、早送り開始インタフェース、早送り解除インタフェース、状態通知インタフェースの 5 つが該当する。ここで提案手法のメッセージの形式ごとの遷移の生成方法を各インタフェースに適応し、状態間の遷移を判断する。

7.3.1.1. モデルの全状態の列挙

モデルの全状態は仕様書に記述されているため、それらを全て状態遷移モデルに列挙する。この仕様書の場合”再生状態”、” 非再生状態”、” 早送り状態”の 3 つが記載されているため、状態遷移図に 3 つの状態を記載する。図 7.3.1.1-1 は全状態を記載したものである。

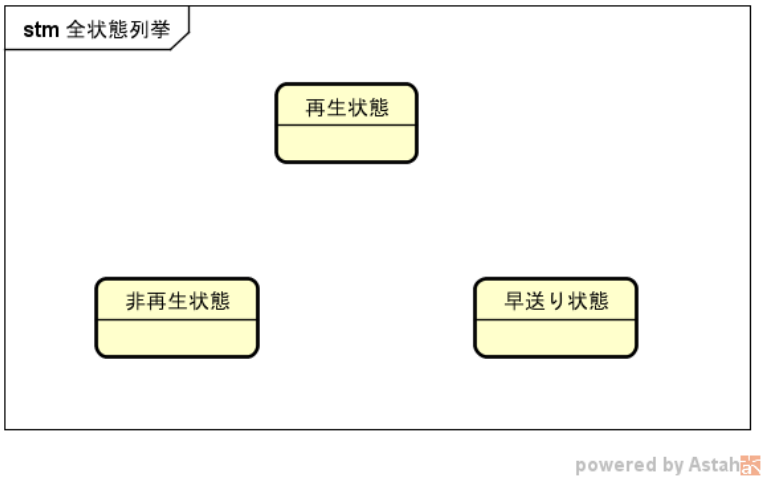

図 7.3.1.1-1 全状態を記載した図

7.3.1.2. 再生開始インタフェースのモデル化

再生開始インタフェースは Request-CallBack 形式であり、Request 側のメッセージとして REQ_PlayBack、CallBack 側のメッセージとして CB_PlayBack がある。先ず、トリガーの要素の情報をメッセージ仕様から抜き出す。トリガーを決めるための情報は「RequestCallBack 形式メッセージのメッセージ名」と「RequestCallBack 形式メッセージのパラメータ」である。このときメッセージ名は”REQ_PlayBack”が当てはまる。パラメータはパラメータ数が 0 であるため設定するパラメータはない。よってトリガーの要素は”REQ_PlayBack”となる。アクション要素の情報は「CallBack 形式メッセージのメッセージ名」、「CallBack 形式メッセージのパラメータ」と「メッセージの説明」から抜き出す。「CallBack 形式メッセージのメッセージ名」は”CB_PlayBack”が当てはまる。「CallBack 形式メッセージのパラメータ」はメッセージ数が 0 であるため、設定するパラメータはない。「メッセージの説明」はパラメータが存在する場合、メッセージのパラメータを決めるために必要な情報であるが、この場合パラメータが存在しないため、「メッセージの説明」の情報は必要ない。よってアクションの要素は” CB_PlayBack”となる。遷移元状態と遷移先状態の要素の情報は「メッセージの説明」から抜き出す。説明の内容を見ると『 非再生状態で本 API を呼び出すと再生開始を要求する。再生開始完了後、再生状態になると CB_PlayBack メッセージを返す。』と記載されており、非再生状態から再生状態に状態が遷移することがわかる。よって遷移元状態は” 非再生状態”、遷移先状態は” 再生状態”となる。以上の内容をまとめると遷移に必要な要素の情報は表 7.3.1.2-1 のようになる。

表 7.3.1.2-1 再生開始インタフェースの遷移を決める要素の情報

遷移を決めるための要素	抜き出し結果
遷移元状態	非再生状態

遷移先状態	再生状態
トリガー	REQ_PlayBack
アクション	CB_PlayBack

再生開始インタフェースの遷移を適応した状態遷移モデルは図 7.3.1.2-1 のようになる。

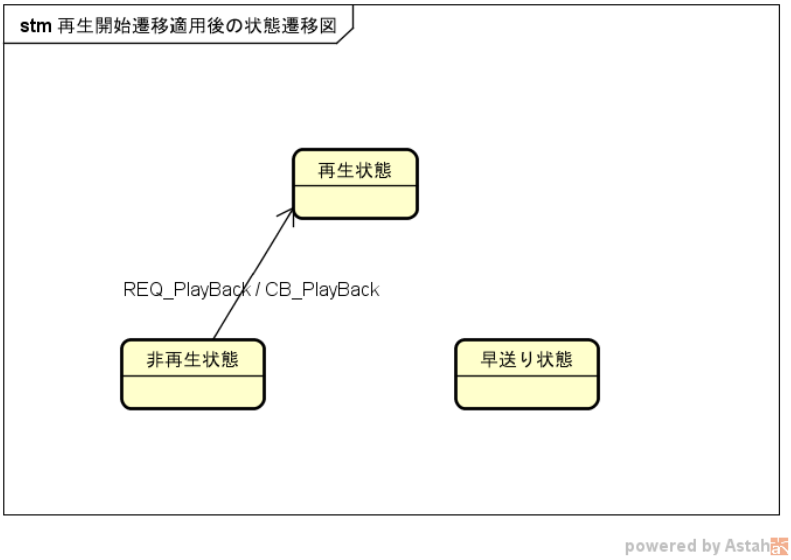

図 7.3.1.2-1 再生開始遷移適用後の状態遷移図

7.3.1.3. 早送り開始インタフェース

再生開始インタフェースと同様に遷移に必要な要素の情報を抜き出すことができる。遷移に必要な要素の情報を抜き出した結果をまとめると、表 7.3.1.3-1 のようになる。

表 7.3.1.3-1 早送り開始インタフェースの遷移を決める要素の情報

遷移を決めるための要素	抜き出し結果
遷移元状態	再生状態
遷移先状態	早送り状態
トリガー	REQ_FastForward
アクション	CB_FastForward

早送り開始インタフェースの遷移を適応した状態遷移モデルは図 7.3.1.3-1 のようになる。

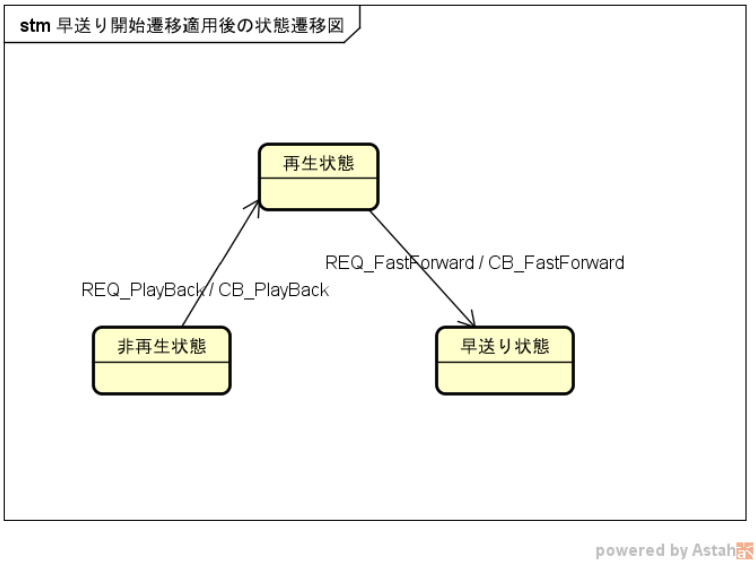

図 7.3.1.3-1 早送り開始遷移適用後の状態遷移図

7.3.1.4. 再生停止インタフェース

再生開始インタフェースと同様に遷移に必要な要素の情報を抜き出すことができる。遷移に必要な要素の情報を抜き出した結果をまとめると、以下の表 7.3.1.4-1 のようになる。

表 7.3.1.4-1 再生停止インタフェースの遷移を決める要素の情報

遷移を決めるための要素	抜き出し結果
遷移元状態	再生状態
遷移先状態	非再生状態
トリガー	REQ_PlayStop
アクション	CB_PlayStop

再生停止インタフェースの遷移を適応した状態遷移モデルは図 7.3.1.4-1 のようになる。

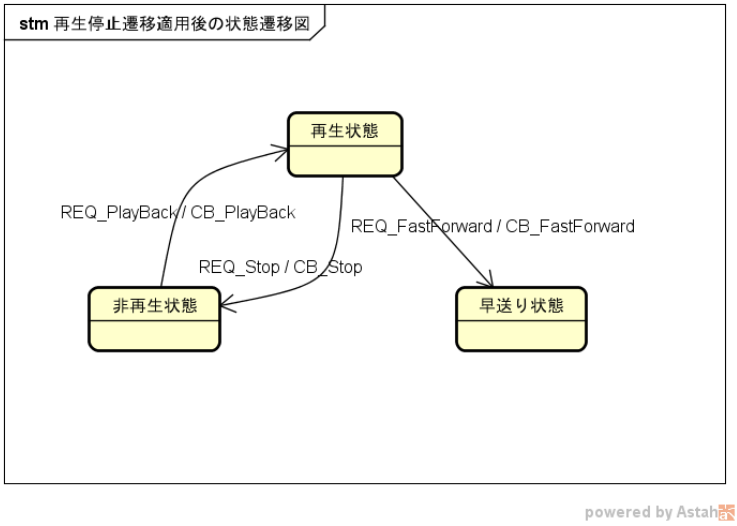

図 7.3.1.4-1 再生停止遷移適用後の状態遷移図

7.3.1.5. 早送り解除インタフェース

再生開始インタフェースと同様に遷移に必要な要素の情報を抜き出すことができる。遷移に必要な要素の情報を抜き出した結果をまとめると、表 7.3.1.5-1 のようになる。

表 7.3.1.5-1 早送り解除インタフェースの遷移を決める要素の情報

遷移を決めるための要素	抜き出し結果
遷移元状態	早送り状態
遷移先状態	再生状態
トリガー	REQ_FastForwardCancel
アクション	CB_FastForwardCancel

早送り解除インタフェースの遷移を適応した状態遷移モデルは以下の図 7.3.1.5-1 のようになる。

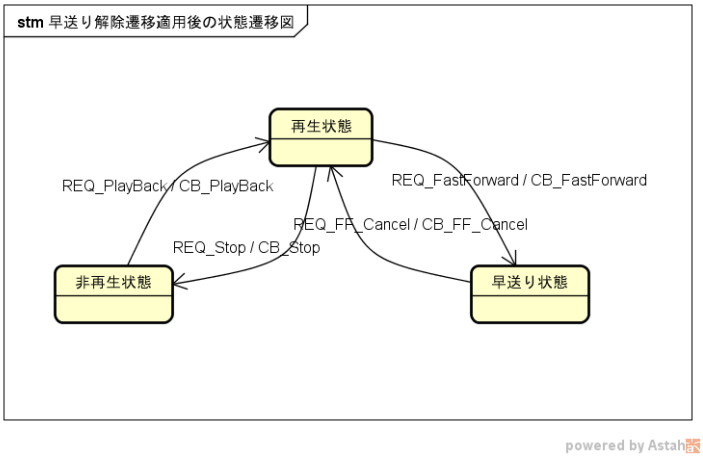

図 7.3.1.5-1 早送り解除遷移適用後の状態遷移図

7.3.1.6. 状態通知インタフェース

状態通知インタフェースは Event 形式である。先ず、トリガーについてだが、Event 形式の場合トリガーはないため、メッセージ仕様から情報を抜き出す必要はない。アクション要素の情報は「メッセージ名」、「パラメータ」と「メッセージの説明」から抜き出す。「メッセージ名」は”Notify_Status”が当てはまる。「パラメータ」はメッセージ数が 1 で型は”MWState”であり、設定可能な値は{Stop, Play, FastForward}である。パラメータにどの値が設定されるかは「メッセージの説明」の項目の情報から判断する。「メッセージ説明」の項目には『MW の現在の状態を通知する。』と記述がある。そのため通知する状態は MW の現在の状態に依存する。遷移元状態と遷移先状態の要素の情報は「メッセージの説明」から抜き出す。ここでメッセージの説明には遷移元状態と遷移先状態の情報が記載されていないため、情報を抜き出すことができない。よって遷移元状態と遷移先状態は”不明”となる。以上の内容をまとめると遷移に必要な要素の情報は表 7.3.1.6-1 のようになる。

表 7.3.1.6-1 状態通知インタフェースの遷移を決める要素の情報

遷移を決めるための要素	抜き出し結果
遷移元状態	[不明]
遷移先状態	[不明]
トリガー	※なし
アクション	MWState{Stop, Play, FastForward} ※現在の状態に依存

遷移元状態と遷移先状態がわからないため、遷移をモデルに反映する場合、取り得る全ての状態を遷移元状態。遷移先状態として掛け合わせた遷移をモデルに反映させる。状態通知インタフェースの遷移を適応した状態遷移モデルは図 7.3.1.6-1 のようになる。

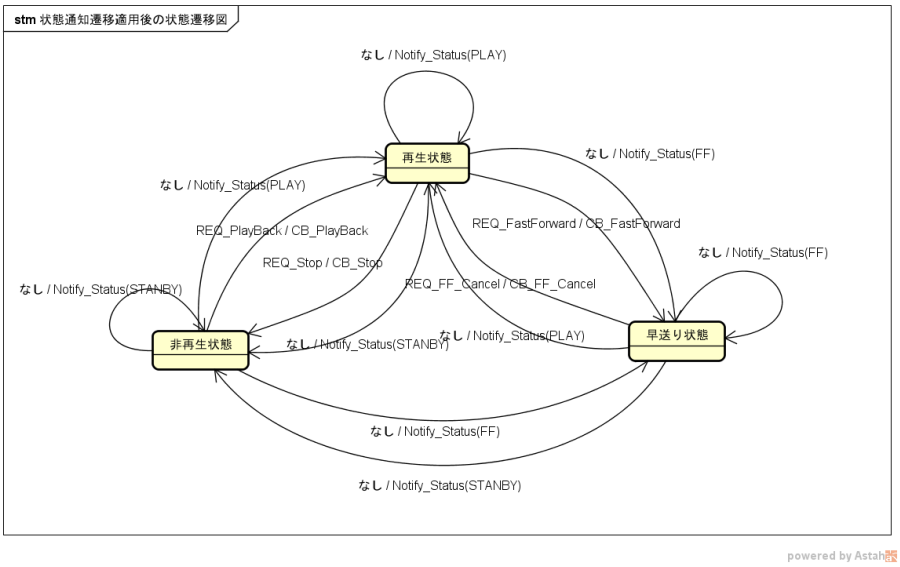

図 7.3.1.6-1 状態通知遷移適用後の状態遷移図

7.3.2. UI 層のモデル化

7.3.2.1. 機器操作の制約のモデル化

UI 層はユーザーの操作の制約を状態遷移モデルとして表す。機器の状態によりユーザーが可能な操作の制約が仕様として決まっているため、それを状態遷移モデルとして表す。機器の状態は「再生停止中、通常速度再生中、早送り再生中」の 3 つがあるため、これが UI 層の状態となる。それぞれの状態で可能な操作と操作後の状態の組み合わせを表でまとめると表 7.3.2.1-1 のようになる。

表 7.3.2.1-1

機器の状態	可能な操作	操作後の機器の状態
再生停止中	再生開始	通常速度再生中
通常速度再生中	早送り開始	早送り再生中
	再生停止	再生停止中
早送り再生中	早送り解除	通常速度再生中

このとき遷移図で表すことを考えると、機器の状態が遷移元状態、可能な操作がアクションであり、UI 層から App 層に対するメッセージ、操作後の機器の状態が遷移先状態になる。トリガーはどの遷移でも”なし”となる。通常速度再生中に可能な操作が 2 つあるが、ユーザーが機器を操作する際にどちらの操作を選ぶかはわからないため、この遷移は非決定的になる。以上をまとめた状態遷移図は図 7.3.2.1-1 のようになる。

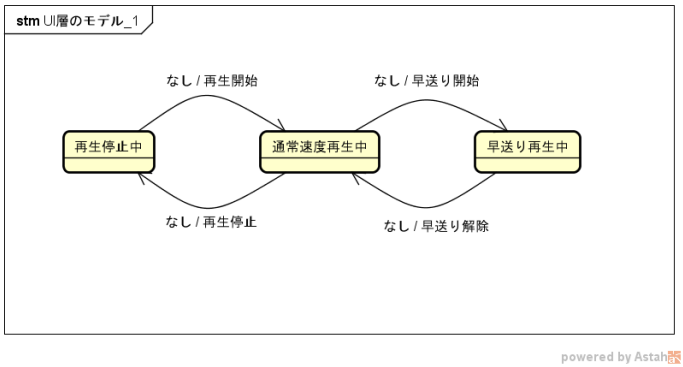

図 7.3.2.1-1UI 層の状態遷移図

7.3.2.2. システムの応答速度のモデル化への反映

ユーザーの操作はそのまま UI 層から App 層へメッセージとして送信される。またそれを受けて App 層から MW 層に対してメッセージが送信され処理がなされる。UI 層、App 層、MW 層それぞれが非同期で動作するモデルの場合、例えば App 層と MW 層のメッセージの送受信が終わる前に UI 層が一方的にメッセージを送る場合がモデル上では起きえる。しかしながら、実際のシステムで考えた場合、UI 層は機器を操作するユーザーの操作にメッセージは対応するのに対し、App 層と MW 層のメッセージ送受信はシステムの間での出来事であるから、UI 層からのメッセージ送信間隔と比べ App 層と MW 層の間のメッセージ送受信間隔は極めて短い。そのため実際のシステムで UI 層からのメッセージが App 層と MW 層の間のメッセージ送受信の間に連続して送信されるようなことは起きえない。よってモデル上でもこのようなケースをなくすようにモデルを変更する必要がある。そこでここではシステムの一連の処理が終わったタイミングで疑似的なメッセージを返し、そのタイミングで次の操作が行われるようにモデルを作成する。具体的にはまず UI 層の状態遷移の中に操作中を表す中間状態を定義する。例えば再生開始の操作が行われ、UI 層から App 層にメッセージが送信されたタイミングで中間状態に遷移させる。この中間状態にいる間は次の操作が行われないようにする。そして再生開始に対応するメッセージ送受信が App 層と MW 層の間で終わったタイミングで、疑似的なメッセージを App 層から UI 層へ送信する。このメッセージを UI 層が受け取ったタイミングで次の状態、この場合は通常速度再生中状態へ遷移するようにする。こうすることで実システム上では起こり得ないような UI 層から App 層へのメッセージ送信をモデルから排除することができる。各操作ごとに中間状態を新たに定義し、疑似的なメッセージを「完了通知イベント」として定義する。新たに定義した中間状態と疑似イベントを含めた UI 層モデルの状態遷移表は表 7.3.2.2-1 のようになる。

表 7.3.2.2-1UI 層モデルの状態遷移表

UI 層モデルの状態	可能な操作	操作後の機器の状態

再生停止状態	再生開始	再生開始待ち状態
通常速度再生状態	早送り開始	早送り開始待ち状態
	再生停止	再生停止待ち状態
早送り再生状態	早送り解除	早送り解除待ち状態
再生開始待ち状態	完了通知	通常速度再生状態
早送り開始待ち状態	完了通知	早送り再生状態
再生停止待ち状態	完了通知	再生停止状態
早送り解除待ち状態	完了通知	通常速度再生状態

7.3.3. App 層のモデル化

App 層では設計時の状態遷移表があるため、状態遷移表がそのままモデルとなる。

7.3.4. App 層と MW 層のメッセージ

App 層と MW 層は API で定義されたメッセージを通じてやり取りされる。そのためモデル上のメッセージでは API で定義されたメッセージが該当することになる。メッセージを App 層から MW 層へのメッセージと MW 層から App 層へのメッセージに分けて一覧にした表を表 7.3.4-1、表 7.3.4-2 に示す。

表 7.3.4-1App 層から MW 層へのメッセージ

メッセージ名
REQ_PlayBack
REQ_FastForward
REQ_PlayStop
REQ_FastForwardCancel

表 7.3.4-2MW 層から App 層へのメッセージ

メッセージ名
CB_PlayBack
CB_FastForward
CB_PlayStop
CB_FastForwardCancel
Notify_Status

7.4. モデルの Promela 記述

7.3 章で作成したモデルをもとに Promela の記述を作成する。Promela 記述ではフォーマルメソッド導入ガイダンス[3]内の要素テクニック：テンプレート支援モデル記述法をもとに作成した。UI 層、App 層、MW 層それぞれがプロセスとなる。プロセスはそれぞれ非同期で動作するため、それぞれのプロセス間通信（UI-App 間、App-MW 間）用に送受信用の入出力チャネルを用意する。各プロセスがとる状態と、送受信のメッセージを mtype で定義する。プロセスの状態遷移は 7.3 章の状態遷移表及び状態遷移モデルをもとに記述する。作成した Promela 記述の一部を図 7.4-1 に示す。

```
/* ステートマシンのための定数 変数の宣言 */
mtype = {
            STATE_STANBY,   STATE_WAIT_PLAY,   STATE_PLAY,   STATE_WAIT_STOP,   STATE_WAIT_FF,
STATE_FF, STATE_WAIT_FF_CANCEL, DUMMY_STATE   /* AppState */

            /* API App - MW */
            REQ_PlayBack, REQ_PlayStop,        REQ_FastForward, REQ_FF_Cancel,      /* Request
*/
            CB_PlayBack,   CB_FastForward, CB_FF_Cancel, CB_Stop,              /* Callback */
            Notify_Status,                                                        /* Notify
Status */

            /* EVENT UI - App */
            EVENT_PLAY, EVENT_FF, EVENT_FF_CANCEL, EVENT_STOP,                 /* Event */

            /* App -> UI Dummy Message */
            EVENT_COMPLETE        /* Complete Message */
        }

/* 各ステートマシンの状態変数 */
mtype UI_state = STATE_STANBY; /* 初期状態 */
mtype APP_state = STATE_STANBY; /* 初期状態 */
mtype MW_state = STATE_STANBY; /* 初期状態 */

/* 各プロセスのイベント受信用通信チャネル */
chan USER_TO_APP_ch = [1] of {mtype};
chan APP_TO_USER_ch = [1] of {mtype};
```

```
chan APP_TO_MW_ch = [1] of {mtype};     // Recieve Callback and Notify from MW
chan MW_TO_APP_ch = [1] of {mtype, mtype};

/* 各ステートマシンの定義 */
/* user model */
active proctype user(){
    do
      :: (UI_state == STATE_STANBY) ->
         printf("*** USER:EVENT_PLAY. ***¥n");
         USER_TO_APP_ch!REQ_PlayBack;
         UI_state = STATE_WAIT_PLAY;

/* ========================================================================== */
/* 以下省略 */
```

図 7.4-1 モデルの Promela 記述の抜粋

7.4.1. 検証内容の設定

検証内容では開発で起きた不具合と同じケースが検知できるように assert を設定した。不具合のケースでは UI 層から App 層に送られたメッセージが状態遷移がうまくいかなかった結果 App 層の状態遷移表上処理が必要ないものとして扱われ、MW 層に対してメッセージが送られなかった。設計者の想定通りに設計されていた場合、UI 層からメッセージに対して何らかのメッセージを App 層から MW 層に送るはずであり、逆にそれがない場合は設計者の想定漏れである。よって assert は App 層のプロセス内でメッセージを受信し、かつその後のメッセージ処理部分の if 文でどの条件にも当てはまらず、else に入ったときに assert の条件に引っかかるようにした。assert 部分の抜粋を図 7.4.1-1 に示す。

```
/* App model */
active proctype app(){
    /* UI_EVENT */
    mtype ui_event;
    /* parameter for receiving notify_status */
    mtype Notify_state;

    do
      /* Receive Event from UI */
      :: USER_TO_APP_ch?ui_event ->
```

```
/* ========================================================================== */
/* 省略 */
/* ========================================================================== */
          :: else -> printf("Illegal State.¥n");
            assert(false);
        fi
```

図 7.4.1-1 assert 部分の抜粋

7.4.2. 検証による反例

生成した promela コードを spin によって検証する。検証することで反例が出力され、これが設計不具合が起こるシーケンスパターンとなる。設計時にはこの反例が出なくなるように設計対象となる App 層の状態遷移表を修正する。修正後にまた検証を実施し、反例がないかを確認する。反例が出なくなるまで App 層の状態遷移表の修正と検証を繰り返した。修正を繰り返す中で潜在する不具合と同じシーケンスパターンの不具合を見つけることができた。

8. 実験の考察

実験ではまず仕様書から手法に沿って状態遷移のモデル化を行った。仕様書では全状態が 3 つインタフェースが 5 つ存在するものを用いた。インタフェースは Request-CallBack 形式が 4 つ、Event 形式が 1 つである。このうち Event 形式には曖昧さが含まれていた。Request-CallBack から状態遷移モデルを作成した結果、遷移は 4 つ生成された。Event 形式では曖昧さが含まれる場合の遷移の反映を行い、この結果として 13 の遷移が反映された。

8.1. 設計者の想定外の振る舞いに起因する不具合を見つけることができる

モデル化の中で曖昧さが含まれる Event 形式の「MW の現在の状態を通知する。」の部分に対し、手法をあてはめることで形式的にモデルを生成することができた。またこれに対してモデル検査を適用することで、不具合を反例として検出することができた。この記述に対して単純に設計した場合では不具合を含んだままになる可能性もあるが、本手法により形式的にモデルを生成することで、曖昧さに起因する想定外の振る舞いをモデル検査の反例としてけとして検出でき、不具合を見つけることができたといえる。

8.2. 半形式的に適用

遷移反映時には分類したメッセージ形式ごとのパターンと曖昧さが含まれた場合の遷移の反映方法を適用することでモデル生成することができ、その後モデル検査により不具合を検出できた。よって半形式的に適用できたといえる。

8.3. 現実的な方法である

今回の実験では実際の開発現場で使われる仕様書の一部を用いて手法の適用を行った。実験では状態 3 つとインタフェース 5 つを用いた。遷移は曖昧さがない部分に関しては 1 つのインタフェースにより 1 つの遷移が生成された。曖昧さがある部分は 9 つの遷移が生成された。実際の仕様書ではインタフェースの数は 40 程度存在するが、曖昧さが含まれる部分については 1 つか 2 つ程度であり、状態も仕様書によるが 10 程度である。そのため実際の仕様書に適応した場合でもモデル生成可能であり、現実的な方法であるといえる。

8.4. 検証時の偽反例

実験時には潜在するパターンの不具合の反例を含め多くの反例が出力された。これは検証用のモデルを作成する際に、仕様書の曖昧さを含めて MW 層のモジュールをモデル化したため、実際のモジュールの振る舞いよりも多くモデルの振る舞いパターンがモデルに反映されているためである。そのため検証時に出る反例には 2 種類存在する。1 つは設計時の状態遷移表の考慮漏れによる反例であり、これは設計不具合である。もう 1 つは実際のモジュールの振る舞いとしては存在しないが、曖昧さを含めたモデル化をしたことによって検証時に出た反例であり、これは偽反例と呼ばれる。偽反例の場合は検証モデルを修正する場合が一般的であるが、本研究手法の場合モデル化対象となる仕様書の曖昧さを含めたモデル化をしているため、偽反例かどうかの判定は難しい。そのため偽反例の可能性もあるがそれも含めて設計不具合として App 層側を修正するようにした。今後の研究課題として偽反例をできるだけなくすために曖昧さを含めてモデル化したモデルからできる限り実際の振る舞いで起きない部分を削る方法を考える必要がある。

9. まとめ

本研究ではブラックボックスなモジュールを含むシステムについての検証手法を提案した。手法ではブラックボックスなモジュールの振る舞いを記述した仕様書に含まれる要素について、実際の仕様書をもとに抽出し、仕様書の一般形について考察した。一般化した仕様書から状態遷移モデルに必要な情報である”取り得る全状態”、”状態間の遷移情報（遷移元状態、遷移先状態、トリガー、アクション）”を抜き出す方法について示した。次に仕様書内の自然言語の記述の曖昧さが含まれる場合にどのように状態遷移モデル生成に影響するかを考察した。その結果状態遷移モデルの遷移情報の遷移元状態、遷移先状態、アクションが正確にわからなくなる可能性があることがわかった。遷移情報が正確にわからなくなった場合のモデル化手法として本研究の提案手法では仕様書の内容から判断して、取り得る全ての可能性についてモデルに反映させるようにした。これによりモデルの検証時にモデルの振る舞いの漏れを防ぐことができる。実験では手法の有効性を示すため実際に開発で起きた不具合を含むオーディオシステムに対して本手法の仕様書からのモデル化手法を適応し、SPIN のモデル検査を実施した。モデル検査を実施した結果事前に含まれた不具合

と同じモデルの振る舞いを反例としてを検知することができ、本手法の有効性を示すことができた。

10. 参考文献

[1] 萩谷昌己, 吉岡信和, 青木利晃, 田原康之, “SPIN による設計モデル検証”, 株式会社近代科学社, 2008

[2] 高田広章編, “ITRON TCP/IP API 仕様”, Ver. 1.00.01” Embedded TCP/IP 技術委員会 （社）トロン協会 ITRON 専門委員会

[3] “フォーマルメソッド導入ガイダンス ～ソフトウェアの安全性・信頼性向上のための技術導入に向けて～ Version1.0”, 株式会社三菱総合研究所, 2011

[4] 中島震, “状態遷移システムデザインのモデル検査”, 2005

[5] 藤原貴之, 岡野浩三, 楠本真二, “SPIN による Struts アプリケーションの動作検証を目的としたモデル生成手法の提案”, 2005

[6] G. J. Holzmann, “The SPIN MODEL CHECKER. Addison-Wesley, 2nd edition”, 2005

[7] 紫合治, “ジャクソン法（JSP）による状態遷移設計”, 情報処理学会論文誌 Vol. 50 No.12 3030-3040, Dec. 2009

[8] Bran Selic, Jim Rumbaugh, “Using UML for Modeling Complex Real-Time Systems”, 1998

[9] Matthew B. Dwyer, Corina S. Pasareanu, “Filter-based Model Checking of Partial Systems”, 1998

[10] Shalini Kaleeswaran, Varun Tulsian, Aditya Kanade, Alessandro Orso. “MintHint: Automated Synthesis of Repair Hints”, ICSE 2014 Proceedings of the 36th International Conference on Software Engineering Page 266-276

[11] Abdelkrim Amirat, Ahcen Menasria, “Automatic Generation of PROMELA code from Swquence Diagram with Imbricate Combined Fragments”, In the Proceedings of 2012 Second International Conference on Innovative Computing Technology, pp.111-116, Sep. 2012

[12] Shing-chi Cheung, Jeff Kramer, “Checking Subsystem Safety Properties in Compositional Reachability Analysis”, Technical Report HKUST-CS95-9, 1995

[13] 楠野明, 岡野浩三, 楠本真二, “シーケンス図が持つメッセージ順序の曖昧性除去手法の提案”, 2015

[14] 安田佳宏, 新川芳行, “時間制約を含む UML シーケンス図の検証手法”, 2010

[15] Edmund M.Clarke, Oma Grumberg and Doron Peled, “Model Checking”, The MIT Press, 1999

[16] David HAREL, “STATECHARTS: A VISUAL FORMALISM FOR COMPLEX SYSTEMS”, Science of Computer Programming 8, 1987

[17] Stefan Leue, Peter B . Ladkin, “Implementing and Verifying MSC Specifications Using PROMELA/XSPIN”, 1991