宮崎大学大学院

博士学位論文

離散事象システムの形式的動作表現モデルとそれに基づく離散型生産システムの系統的運用・制御方式の提案

2013 年 12 月

宮崎大学大学院農学工学総合研究科

博士後期課程

物質・情報工学専攻

高塚　佳代子

目次

記号の説明

本論文で使用した記号の意味を以下に示す。

略語

ETKS（Extended Timed Kripke Structure）	：拡張時間 Kripke 構造
ETSC（Timed State Chart）	：拡張時間ステートチャート
IDJ（Identification of Job）	：ジョブ識別子
PNM（Process Network Model）	：プロセスネットワーク型モデル
R-ETSC（Restricted Timed State Chart）	：制限された拡張時間ステートチャート
TKS（Timed Kripke Structure）	：時間 Kripke 構造
TSC（Timed State Chart）	：時間ステートチャート

離散事象システムの形式的動作表現モデルとモデルに基づく離散型並列生産システムの系統的運用制御方式

2013 年 12 月

宮崎大学大学院農学工学総合研究科

物質・情報工学専攻

高塚　佳代子

概要

離散型並列生産システムでは、多数の逐次操作や並列操作が階層的かつ事象駆動で実行されているが、その複雑な動作を明快に表現できる挙動表現モデルは未開発であり、そのことが体系的な制御系設計手法や動作検証手法開発の遅れの要因となっている。本論文では、まず、固有の複雑さを有するこの種のシステムの挙動表現モデルを開発し、次にそれに基づいて「不確定性のある離散型並列生産システムの系統的制御系設計法」を提案した。また、この挙動モデルを用いて、実システム稼働時に与えられる運用戦略や制御則の妥当性検証を、モデルチェッキング型検証法の枠組みで行う方法を与えた。以下、その概要を示す。

第 1 章は序論であり、本研究の背景、当該分野での現状と課題、および、本研究の目的を述べた。

第 2 章では、離散型生産システムの動作表現モデル、制御系設計法、動作検証法に関する既存研究の紹介と、本研究でベースモデルとした「時間ステートチャート」の概要およびそれが持つ制約について述べた。

第 3 章では、離散型並列生産システムの複雑な挙動を明確に表現でき、動作解析にも使用できる挙動表現モデル「拡張時間ステートチャート」を開発した。具体的には、実時間並行ソフトウェア検証用モデルである「時間ステートチャート」に対し、離散型生

産システム固有の挙動表現に必要となる種々の拡張（イベント生成機能、発火可能期間制約、複数ジョブ識別用ジョブＩＤの導入）を行うとともに、同モデルの動作的意味を規定する「イベントプール」と「管理マネージャ」を新規に導入し、同モデルを動作解析にも使用可能なものにした。

第 4 章では、「離散型並列生産システムのモデルに基づく系統的制御系設計法」を提案した。 この種の生産システムは基本的に事象駆動であり、各操作の所要時間は不確定に変動する。そのため、実行系でこの変動に対してロバストな制御を実現するには、計画系から与えられた目標スケジュールに近い動作をする事象駆動型の制御系を別途作成して与える必要があり、そこでは人手の介在が不可避となっている。本章では、このような計画系・実行系間での不連続性が無く、上記の不確定変動にロバストな制御系の実現方法を提案した。具体的には、対象プラントの構造を反映したプロセスネットワーク型の中間モデルを介して挙動モデルを作成する方法を与えてモデル作成上の個人差を無くすとともに、作成されたモデルから「制御に必要な動作と情報」のすべてを５つのテンプレートを用いて系統的に抽出する方法を考案して、スケジュール切替えの際にもプラントを止めることなく連続運用できるような制御系設計の枠組みを与えた。さらに、本手法に則してミニＦＡプラントの制御系を実装し、本実現方法の有効性を実機実験により確認した。

第 5 章では、上記の「遅れ」に対するロバスト性の限界評価にモデルチェッキング型検証手法を適用できるようにするための検証モデル(時間Ｋｒｉｐｋｅ構造)の拡張を行うとともに、現実規模のプラントでは常に問題となる「計算量の爆発」への対処法として、不確定性を有限個の非決定性で置換可能な対象に対する実用的な検証アルゴリズムを与え、標準的バッチプラントモデルへ適用し、同検証手法の有効性を示した。

第 6 章では、本研究のまとめと、残された課題、今後の展望について述べた。

第１章　序論

1.1　はじめに

離散型並列生産システムでは、階層化された多数の逐次動作や並列動作が、事象駆動で実行されている。しかし、離散型並列生産システムは、リアクティブ性、並行性、非決定性、操作上の自由度の多さといった複雑な挙動を共通に持っており、そのような挙動を明快に表現できる適切な動作表現モデルが未開発であることが、この種のシステムの制御系設計や動作検証での体系的方法論開発遅れの要因となっている。以上のようなシステムを効率的に運用するためには、適切な運用戦略や制御システムを与えることが特に重要であり、このような運用戦略や制御方法の妥当性をチェックするためには、システムの振る舞いを的確に表現できるようなモデルの存在が必須である。

一般に、非同期分散システムに分類されるこの種のシステムのためのモデリング手法としては、プロセス代数、時相論理、ペトリネット、半語・半言語、時間オートマトン、Promela、Function Block (IEC61499)、時間ステートチャート(TSC)、などが研究されている [1)~17)]。しかし、いずれも、離散型並列生産システムを想定した手法ではないため、そのまま利用できるモデルは見当たらない。

上記モデルのなかのTSC[1),2)]は、航空機制御システムや計算機のオペレーティングシステムのような実時間並行ソフトウェアの仕様記述モデルとして開発されたモデルであるが、離散型並列生産システムの動作表現モデルとしての要件をもっとも多く満たすモデルである。そこで、本研究では、当初、離散型並列生産システムの挙動表現モデルとしてTSCを利用できないかと考えた。しかし、実時間並行ソフトウェアで示されるシステムの並行性と、離散型並列生産システムでの並行性とでは、その内容が大きく異なる。まず、実時間並行ソフトウェアが示す並行性とは、一つのジョブの実行が多くのプロセスの並行動作によって実現されるという並行性である。そのため、そこで管理されるべきより本質的な制約条件とは、動作が正しいタイミングで行われるかどうかである。そのため、並行プロセス間のタイミング制約がより本質的な制約となる。一方、本研究で扱う離散型並列生産システムでは、一つのプラント上で、同時並行的に処理される複数のジョブを所定の時間通り(納期内)に完了させることが目的である。そのため、そこで管理されるべきは、各プロセスを実行するための各装置上でのジョブ同士の競合であり、このような競合を、各イベントの発火可能期間を満たしながら解消するための制御が問題となる。従って、各イベントの発火可能期間の制約がより本質的な制約条件となる。以上が、離散型並列生産システムのためのモデルを新たに開発する必要がある理由である。

上述の「挙動表現モデルが存在しない」ことから生じている大きな問題として、離散型並列生産システムの制御系設計を系統的に行うための方法論が未開発であるということがある。具体的には、制御系設計を系統的に行うためには、核となる

ベースモデルの存在が不可欠であるが、そのようなベースモデルを正確に記述できる挙動表現モデルが存在しないということである。そのため、生産プラントを扱う各企業では、制御プログラムの作成・追加・変更など、独自に蓄積してきたノウハウに基づき、かなりの手作業を介してプラント運転しているのが現状である[18)~21)]。

また、生産システムでは、処理時間の不確定性や操作上の非決定性があるため、これらに因って生じる(生産計画からの)ずれに「システムが何処まで耐えられるか」といった適応限界を調べる検証が必要である。非同期分散システムのこのような検証をある程度厳密に行うための方法としては、モデルチェッキング型の検査方法[3),4)]が有効であるが、そのためにも、挙動表現モデルが必要である。しかし、上述してきたように、生産システムの振る舞いを的確に表せる挙動表現モデルはなかったため、これまでは、この種のシステムの動作検証では、テストデータを使ったシミュレーション程度のチェックしか行われてこなかったというのが現状である[22)]。

1.2 本研究の目的

以上のことを踏まえ、本論文の目的は、まず、離散型並列生産システムの複雑な動作を的確に表せるような挙動表現モデルを提案することである。そして、そのモデルに基づいた制御系設計のためのフレームワークを提案することである。更に、離散型並列生産システムの適用限界を調べるための検証方法として、提案モデルを使ってできるモデル検査型の検証手法を開発することである。具体的な研究課題は、以下の通りである。

（1）離散型並列生産システム特有の複雑な振る舞いを簡潔に表現し動作解析できる形式的モデルを提案すること。具体的には、まず、離散型並列生産システムの「動作的特徴」を挙げ、これら動作的特徴を持つ生産システムのモデル化に必要な要件を元々最も多く備えている既存モデル「時間ステートチャート」(TSC)をベースモデルとして採用し、更に、その TSC で扱えない動作的特徴(複数ジョブの並列実行、制御方法の例外時の切り替え等)を扱えるように TSC を拡張すること。

（2）提案する挙動表現モデル「拡張時間ステートチャート」(以下“ETSC”)をベースモデルとして使用し、処理時間の不確定性に対してロバスト性を持つ「事象駆動型生産システム制御系のモデルに基づく設計のためのフレームワーク」を提案すること。

具体的には、まず、記述上の個人差を無くす目的で、ETSC を限定したサブセット「R-ETSC」を導入する。そして、時間駆動の情報として得られる目標スケジュールを、処理時間が多少変動しても利用可能な事象駆動の情報へ、系統的に受け渡す方法を提案する。具体的には、まず、制御に関する情報を、スケジュールの変更によって動的に変わる部分と、スケジュールによって変わらない静的な部分とにモデルベースで切り分けて記述できるような仕組み(テンプレート)を導入する。また、これ

らテンプレートに埋め込む情報を、R-ETSC によって記述されたシステムの振る舞いからシステマティックな手順で抽出する方法を示す。更に、以上のようなテンプレートを利用する方法を使って、より大きな外乱により再スケジューリングが不可避となった場合においても、現在稼働中の対象プラントを止めることなく、その再スケジューリング結果を取り込み、スムーズにシステム運用を継続できるような制御の実現方法を示す。

（3）上記アプローチが不確定性を有する事象駆動型システムのロバストかつ継続的な連続運用制御に有効であることを、ミニ FA プラントを対象とした検証実験により確認すること。

（4）離散型並列生産システムの適応限界を調べるための検証方法として、本稿で提案するモデル ETSC に基づくモデルチェッキング型の検証方法を提案すること。具体的には、この種の検証手法、即ち、「Kripke 構造全体を時間領域分割して状態ノードを作成しその上での検証を行う検証手法」が元々持っている計算量爆発の問題への対処方法を考案・導入することにより、モデル検査型の検証手法を ETSC に適用可能にする。

（5）Japan Batch フォーラムが作成したバッチプラントの標準モデルへの適用実験を通し、上記（4）のアプローチがこの種のシステムの検証に有用であることを確かめること。

1.3 本論文の概要と構成

本論文は、第 1 章から第 6 章で構成されている。

第 1 章は序論であり、本研究の背景と、系統的制御系設計法および論理的動作検証法の開発における現状と課題を示すとともに、本研究の目的について述べた。

第 2 章では、離散型並列生産システムのための挙動表現モデル、制御系設計法、及び動作検証法の開発に関連する既存の研究を紹介するとともに、挙動モデル開発でベースモデルとした「時間ステートチャート」の概説と、拡張が必要となる理由について述べる。

第 3 章では、この種の生産システムの複雑な挙動を簡潔に表現でき、動作解析にも使用できる形式的モデル「拡張時間ステートチャート ETSC」を開発導入する。具体的には、実時間並行ソフトウェアの動作検証用に開発された「時間ステートチャート」をベースモデルとし、それに離散型生産システム固有の振る舞いを表現するために必要となる種々の拡張（内部イベントの生成機能、発火可能時間制約の導入、複数ジョブの同時並列実行識別のためのジョブＩＤの導入等）を行う。また、同モデルの動作的意味を規定する「イベントプール」とその「管理マネージャ」を導入し、同挙動表現モデルを動作解析にも使用できるようにする。

第 4 章では、上記の挙動表現モデルを用いた「離散型生産システムのモデルに基づく系統的制御系設計のフレームワーク」を提案する。 この種の生産システムは一般に事象駆動で動作しており、各処理の所要時間は不確定に変動する。そのため、実行系でこの変動にロバストな制御を実現するためには、計画系から時間駆動の形で与えられた目標スケジュールを参照して、それに近い動作をする事象駆動型の制御系を構築して与える必要があり、その作業には人手の介在が不可避となっている。本章では、このような計画系・実行系間での不連続性の無い、ロバストな制御系を実現する方法を与える。具体的には、対象プラントの構造を反映したプロセスネットワーク型の中間モデルを ETSC モデルに変換することで挙動モデル作成上の個人差を無くすとともに、「制御に必要な動作および情報」のすべてを当該挙動モデルから系統的に抽出する方法を考案、実行時にはマルチスレッドで動作する実機駆動部にそれを受け渡すことにより、望む制御を実現する、という方法を提案する。同時に、再スケジューリング時などのスケジュール切り替えの際にも、プラントを止めることなく運用を継続できるような仕組みも実現している。さらに、ミニ FA プラントを対象として制御系を実装し、不確定性を有する事象駆動型システムのロバストかつ継続的な運用制御システムの実現に有効であることを、実験により確認している。

第 5 章では、上記の不確定性や非決定性によって生じる「遅れ」に対するロバスト性の限界評価にモデルチェッキング型の検証手法を適用できるようにするための拡張を行う。また、現実規模のプラントでは常に問題となる「計算量の爆発」への対処法として、不確定性を非決定性で置き換えられるような対象に対する実用的な検証アルゴリズムを与え、標準バッチプラントモデルへ適用、本手法がこの種のシステムの動作検証に有効であることを示す。

第 6 章では、本研究のまとめと、残された課題、今後の展望について述べる。

第 2 章 離散型並列生産システムの挙動表現モデルの現状

2.1 離散型並列生産システムとは

本稿において、“離散型並列生産システム”とは、一般的な“離散システム”や“並列システム”の性質を持つような生産システムという意味で用いる。具体的には、以下の通りである。

生産システム： システムとは、JIS の定義において、「多数の構成要素が有機的な秩序を保ち、同一目的に向かって行動するもの」である。製造業において生産という目的を達成するための統合された仕組みは、まさにこの定義に当てはまる。即ち、生産システムは、要素としては、人々と、製造機械と、作業空間（建築）と、そして情報をやりとりするための仕組み（IT システムや帳票類）から成り立っている。また、その生産システムの機能とは、「需要情報というインプット(生産要求など)を、製品というモノに変換してアウトプットする」ことであり、そのインプットからアウトプットまでの時間(スループット)を出来るだけ短縮することが求められる。また、副次的なインプットとして、原料・部品と用役・副資材などを利用する[23)]。

離散システム： システムを記述する変数（入力，出力または状態等）が時間または空間に対して離散的に振る舞うシステムである。

並列システム： 協調動作する複数のプロセスから構成されるシステムである。

2.2 離散型並列生産システムの例

離散型並列生産システムの例として、ネットワーク型生産システム、組み立て式プラント、バッチ式プラントなどがある。

ネットワーク型生産システム

ネットワーク型生産システムとは、地理的に離れた複数の生産拠点による分散・ネットワーク型の生産システムである。このようなシステムでは、生産要求の変化や緊急事態の発生に柔軟な対応をするため、集中管理方式ではなく、分権・協調的な運用が必要である。そのため、関係する各部門の間で迅速かつ正確な情報伝達を行えるような情報交換の手段や仕組みが必要である[24)]。

Fig.2.1 はネットワーク型生産システムの一例として、実在の特殊プラスチックフィルム製造プラントの（1）概観、（2）各拠点が保有する装置の種類、（3）製造工程、を表している。本対象システムの特徴は、以下の通りである[25)]。

・地理的に分散した複数の生産拠点と１つの生産計画部門とからなる。

・生産計画部門は、注文をとり最終製品を生産する各拠点へ生産量を割り当てる。

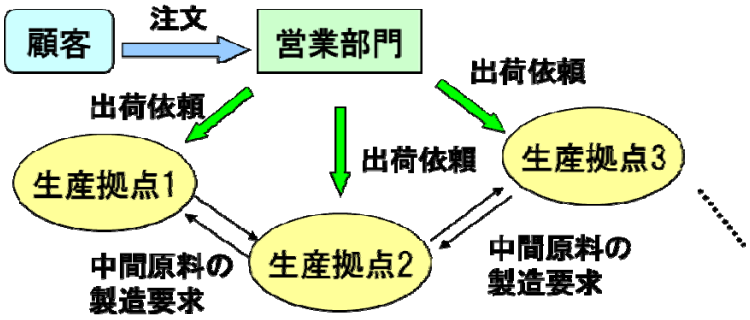
・製造工程は高々３工程で、扱われる製品は１拠点のみでは完結しないようなものが含まれている。

・各拠点は、割り当てられた最終製品を期日までに製造するための計画立案から製造活動までを独自に行う。そこで必要となる材料(原材料/中間製品)の調達先の決定も独自の判断で行なう。

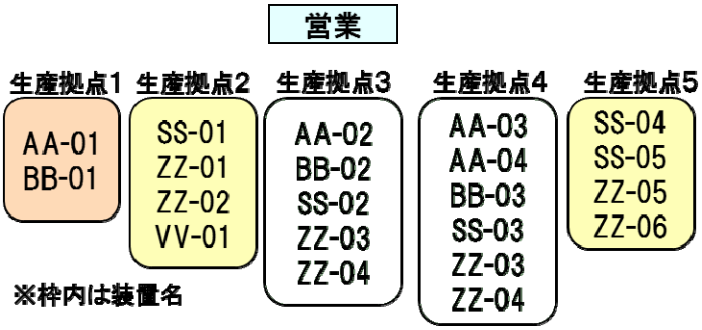
・本生産システムでは計画の修正や変更が頻繁に起きる。

以上のようなネットワーク型生産システムは、柔軟性確保の観点から、集中管理ではなく分散協調により運用されるため、原材料の調達計画などでは多重の時間遅れが生じるという難しさがある。

（1）プラスチックフィルム製造プラントの概観



（2）各生産拠点の保有する装置



（3）製造工程

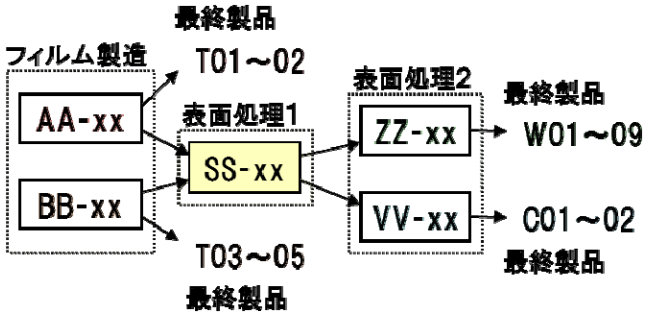


Fig.2.1: ネットワーク型生産システムの例 （プラスチックフィルム製造プラント）

組み立て式プラント

組み立て式プラントとは、ライン生産方式を取る生産プラントである。ライン生産方式とは、大量生産を行う工場で製品の組み立て工程、作業員の配置を一連化（ライン化）させ、ベルトコンベアなどにより流れてくる機械に部品の取り付けや小加工を行う作業方式である。この方式は、ヘンリー・フォードが創始したフォード・モーター社（1903 年創設）において実施した経営管理方式であるため、別名、「フォード・システム」と呼ばれる。「（少品種）大量生産」を基本コンセプトとし、「標準化」と「移動組立ライン」という大きく二つの要因がこの生産システムを支えるということが、ライン生産方式の特徴である。以上のライン生産方式に対し、近年は、市場のニーズの多様化を受け、従来の量産効果を発揮しながらも、多品種多仕様化にも適応しようとする生産方式「多品種組立ライン」という方法が編み出されている。この方式には、バッチ式組み立てラインと混合品種組み立てラインの２種類がある。具体的には、バッチ式組み立てラインとは、ある期間中に生産される品種数に応じて、計画期間をいくつかの小期間に細分し、細分された小期間は一品種のみを連続的に生産する方式のことである。また、混合品種組立ラインとは、作業方法がほぼ等しいので特定の複数品種を混合して連続的に生産するために、あらかじめ準備された一本の生産ラインである複数品種を混合して連続的に生産する方式のことである。[26),27)]

Fig.2.2 は、組み立てラインの一例である、Factory automation のミニチュアの実験装置の写真と模式図である。この実験装置は、ターンテーブルを第１工程、ストッカーを第２工程とする、２工程のフローショップ型生産システムに見立てたものであり、それぞれ４つの加工装置がある。パーツフィーダ上の処理対象は、ピック＆プレースによって搬入コンベア上に運ばれ、アームロボットによってターンテーブル上の装置Ｔ１～Ｔ４のいずれかに運ばれる。ここで第１工程の処理が行われ、一定時間経過後に、再度アームロボットによって搬出コンベア上に運ばれる。搬出コンベアでストッカーまで運ばれてきた処理対象は、押出シリンダーによって任意のストッカーへ送り出される。ここで第２工程の処理が行われ、一定時間経過後に製品が流し出される。この実験装置で制御すべき事柄は、ターンテーブル上の４つの装置のどれに処理を行なわせるか、使用が競合するアームロボットの使用順序(ワークの搬入コンベア端からのターンテーブル上への搬入、及び、ターンテーブル上の処理装置から搬出コンベア端へのワークの運び込み)等が問題となる。以上のような装置割り当て順序は、実行前に生産スケジュールを行い、準最適解を予め与えることが多い。しかし、第１工程、第２工程共に、処理時間の不確定性があるため、与えたスケジュールの準最適性が保たれる保障はないという難しさがある[28)~34)]。

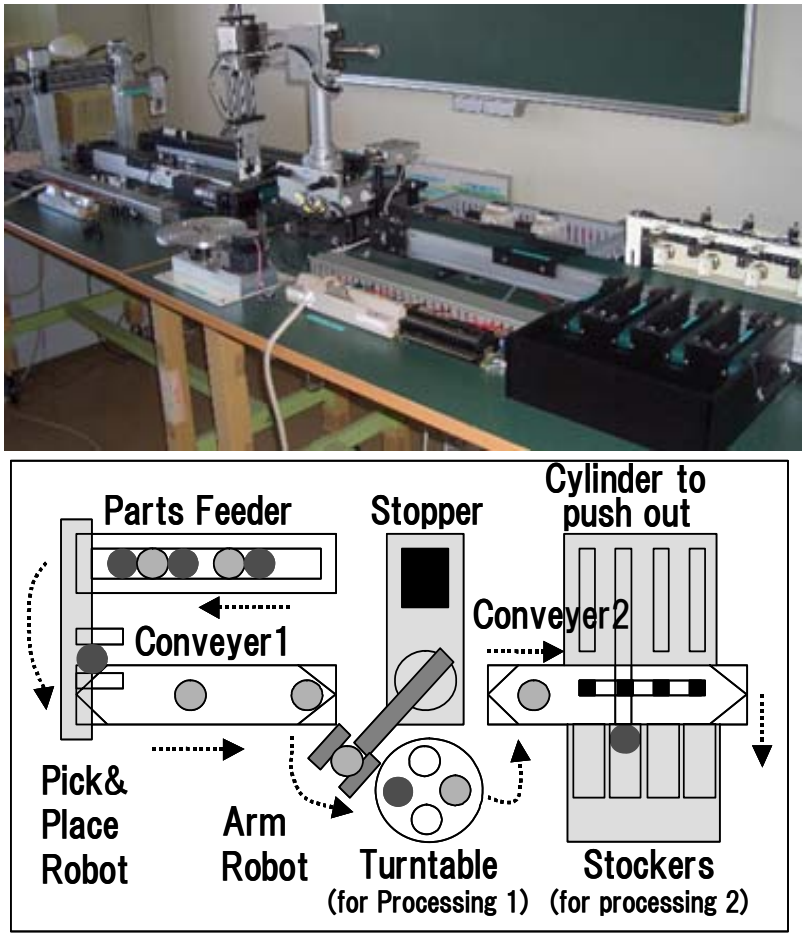


Fig.2.2: 組み立てラインの例 (Factory Automation)

バッチ式プラント

バッチ式プラントとは、パッチ式の処理形態を取るプラントのことである。バッチ処理とは、ひとつの設備である程度まとまった時間、または単位操作ごとに処理を区切り、原材料をこの区切りごとにまとめて投入する処理のことである。発酵工程やバッチ殺菌などは典型的なバッチ処理である。バッチ処理の特徴は、それぞれの工程にかける時間やその他の条件の変更が任意に行えるため、多品種少量生産に適していることである。なお、バッチ処理の反対語は遂次処理、連続処理であり、その特徴は、バッチ処理と反対に、少品種大量生産に向き、多品種少量生産には不向きである[21)]。

Fig.2.3 はバッチ式プラントの一例である、化学プラントの概観である[22)]。本プラントは、同等の機能を持つ A, B 2 つの系列から成る二段重合反応バッチプラントである。製造工程は、第一重合缶において原料タンク 0,1 より原料を仕込み、温調と攪拌を同時に並行して行うという処理を、品質が一定値を示すまで繰り返し行い、窒素加圧、静置を経た後に第二重合缶へ全て払い出すという流れとなる。また、第二重合缶に関しても、同様の製造工程を持つ。なお、本プラントの原料タンク 1,2 は配管を共有

しているため、A 系列と B 系列の重合缶への同時供給はできないという制約が与えられているため、これらのタンクを使用する際に競合が発生する。従って、原料供給の競合を解消するための何らかのルールが必要である。以上のような競合解消のためのルールは、実行前にスケジューリングを行い、その結果を準最適なルールとして与えることが多い。しかし、反応缶での反応時間等に不確定性があるため、与えたスケジュールに従い続けることで準最適性が保たれるという保障はないという難しさがある。

また別なバッチ式プラントの一例である Fig.2.4 は、JBF(Japan Batch Forum)で作成されたバッチ式化学プラントの標準モデルの概観である [35]。本プラントは、４階建て構造のプラント全体は独立な 12 の装置、即ち、原料調整缶 A,B(4F)、滴下缶 A,B(4F)、反応缶 A,B(3F)、晶析缶 A,B(2F)、濾過機 A,B(2F)、乾燥機 A,B(1F)の 7 種 14 個の独立な装置から成る。また、製造過程では、最上階の装置から下層の装置へ順に中間原料を送りながら、製品を加工していく。即ち、製造過程での装置使用順序は、溶媒タンク，滴下缶，原料調整缶→反応缶→晶析缶→濾過機→乾燥機である。なお、反応缶(3F)以降の工程において、たすきがけ運転が可能、即ち、A 系列から B 系列、或いは B 系列から A 系列への原料移送が可能である。また、追いかけ運転可能、すなわちプラントは、次々と製造要求が発生する場合に、前の製造要求に対して稼動中であっても、空き状態の装置に次の製造要求を処理させること可能である。以上のことから、反応缶での処理以降は、A 系と B 系のどちらの装置を使用するかに関する装置使用上の非決定性がある。このような非決定性への対応は、事前の生産スケジューリングの結果に基づき、確定させることが多い。しかし、反応缶での処理時間や、乾燥に掛かる時間等、処理所要時間の不確定性の問題があるため、与えたスケジュールに従い続けることで準最適性が保たれるという保障はないという難しさがあるということは、先の例題と同様である。

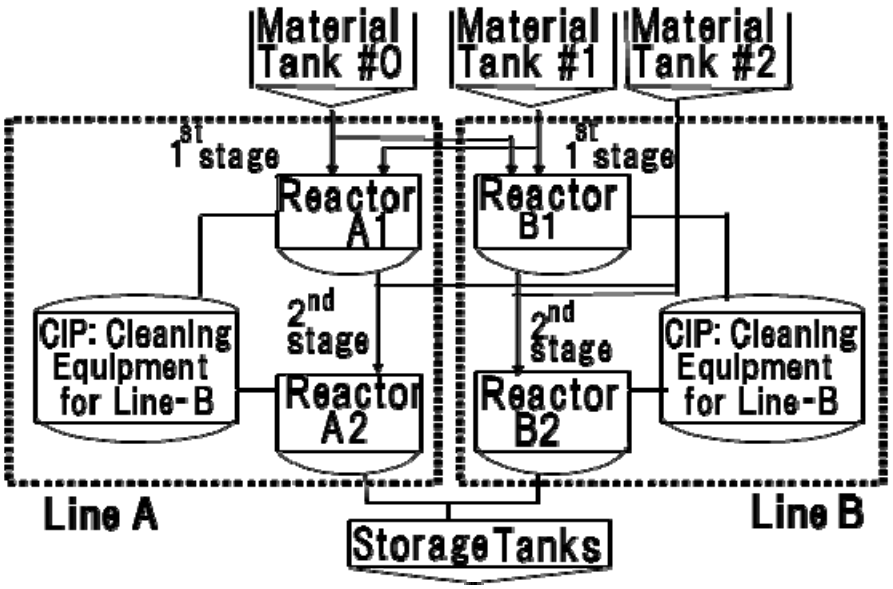


Fig.2.3: バッチ式プラントの例-(1)

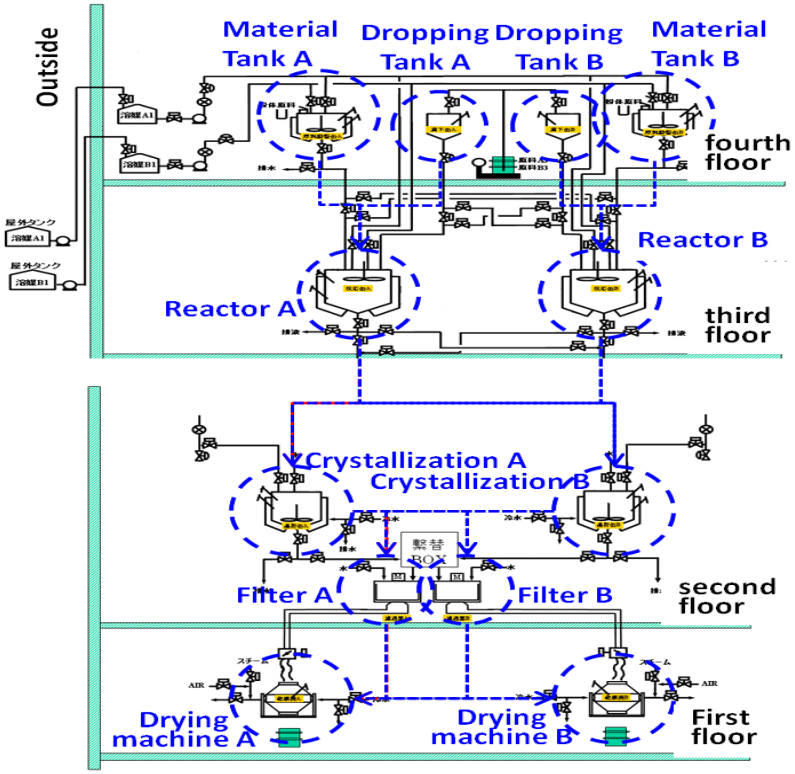


Fig.2.4: バッチ式プラントの例-(2)

以上の例から分かるように、離散型並列生産システムは、並行性、非決定性、操作上の自由度の多さといった問題を共通に持っているシステムである。また、離散型並列生産システムは、リアクティブ性の問題を持つ“リアクティブシステム”であることも一般的に知られている。リアクティブシステムとは、直感的には、「動作している間に、外界から刺激(入力)を受けると、その入力がシステムの与える制約を満たすものであれば、必ず、それに対する応答を返す」といった振る舞いを繰り返すシステムのことである。システムがこのようなリアクティブ性を持つ場合、そのシステムの妥当性を言うためには、非常に状態数の多いシステムのあらゆる実行の可能性を確かめる必要がある。

以上のように、扱いの難しさの原因となる性質を複数持つ離散型並列生産システムを効率的に運用するためには、適切な運用戦略や制御システムを与えることが特に重要である。そして、このような運用戦略や制御方法の開発や、その妥当性をチェックを行うためには、システムの振る舞いを的確に表現できるようなモデルの存在が必須である[36)~43)]。

2.3 並列性・分散性・リアクティブ性を持つシステムのためのモデリング手法について

離散型並列生産システムが持っているような、並列性、分散性、リアクティブ性等の性質を持つシステムのためのモデリング手法としては、プロセス代数、時相論理、半語・半言語、ペトリネット、時間オートマトン、時間ステートチャート、時間ステートチャート(TSC)、Function Block (IEC61499)、などが研究されている[1)~17)]。

プロセス代数

プロセス代数とは[8)]、計算機科学において並行システムを形式的にモデリングする CSP,CCS 等の各種手法の総称である。プロセス代数では、独立したプロセス間の相互作用を、共有変数の更新ではなく通信（メッセージパッシング）として表現する。また、プロセスやシステムの記述には、少数のプリミティブ(SKIP：何もせずに正常終了する，STOP：何もせずに異常終了する，という特殊なプロセス）とそれらプリミティブを組み合わせた演算子を使うプロセス演算子の代数学的規則を定義しプロセスの表現を方程式を解くように操作可能とする。プロセス演算子としては、以下のようなものがある。

・チャネル出力： chan!e （e:イベント，chan:チャネル，chan を介した e の出力）

・チャネル入力： chan?v （e:イベント，chan:チャネル，chan を介した e の入力）

・Prefix イベント： a → P （a:アクション，P:プロセス，最初に **a** を実行し，次に **P** と同じ振る舞いをするプロセス）

・遂次処理： P;Q (P:プロセス, Q:プロセス)

・並列処理： P || Q (P:プロセス, Q:プロセス, P と Q の並行実行を表すプロセス)

・外部選択： P □ Q (P:プロセス, Q:プロセス, P と Q の並行実行を表すプロセス)

・内部選択： P ⫿ Q (P:プロセス, Q:プロセス, P または Q の実行を表すプロセス)

・インターリーブ： P ||| Q (P:プロセス, Q:プロセス, P と Q の完全に独立した並行動作)

・隠蔽： P＼X (X:イベントの集まり, P:プロセス, P からイベント X を隠蔽して得られるプロセス)

以上のように、プロセス代数ではプロセスを代数式で表記させるため、代数的な計算が可能であり、正当性の証明が計算で出来るという特長を持つ。そのため、検証を考慮した仕様記述手法としては望ましい。また、プログラミング言語との親和性が高い、コンポーネント(何らかの機能を持った、プログラムの部品)を組み合わせていくような合成が容易、といった利点もある。ただし、図的表現を持たないため、対象の概観や振る舞いを直感的に理解することは難しい。

時相論理

時相論理とは [5)~7)]、古典的な命題論理(PL: Propositional Logic)に、必然, 可能, などの様相をあらわす論理記号を追加した様相論理のなかでも、特に、時間の様相を持つ論理である。LTL: Linear-Time Temporal Logic (線形時間時相論理)と CTL: Computation-Tree Logic (計算木論理)(或いは、BTL: Branching Time Temporal Logic(分岐時相論理))とがある。

まず、LTL では、次の時刻に“世界”(或いは“可能世界”)が取り得る状態は唯一に決まる、という線形な時間構造に基づいて、通常の命題論理に時相記号{X,F,G,U}を付加することで拡張する。

Xφ (next): 次の時刻で- が真

Gφ(globally): これから先ずっとφが真

Fφ(future): 将来のある時点でφが真

φU (until): 将来のある時点で が真で、その直前までずっとφが真

一方、CTL(或いは BTL)では、次の時刻において可能世界がとり得る状況に複数の可能性があるものと考え、PL を拡張したものである。具体的には、記号 {A,E} を先の LTL に追加する。

Aφ(all): 次の時刻のすべての可能性でφが真

Eφ(exists): 次の時刻のある可能性でφが真

また、時相論理の式は、例えば、LTL の場合、以下の規則を有限回適用することで構成される。

(1)PL の論理式は LTL の論理式である。

(2)p,q が LTL の論理式ならば、p，q に LTL の論理記号を施して得られる式（ p∧q，¬p，Xp，Fp，Gp，pUq ）は LTL の論理式である。

なお、CTL 式も同じ考え方で構成される。

以上のように、命題論理式では、ある状態の真偽しか記述することができないが、時相論理式ではある状態から次状態でどう変化したか、いつまでその状態が成り立つか等を記述することができる。そのため、ハードウェアやソフトウェアの時間に依存して起こる事象の記述に適している。仕様を時相論理式で記述すると、定理証明アルゴリズムでの検証や、モデルチェッキングによる検証が可能であり、プロセス代数同様、検証を考慮した仕様記述手法としては望ましい。しかし、動作の結果生じる各状態を時相論理式として扱うため、動作が理解しにくい。

半語・半言語

半語はトランジションを要素とする半順序多重集合であり、半語の集合を半言語という[9),10)]。事象の半順序構造、及び、半順序構造間の並行動作を陽に表現できる手法であり、プレースの容量が 1 である条件／事象ネット(C/E ネット)の動作解析のための手法として用いられることが多い。与えられた C/E ネットが示し得る振る舞いの全てを半言語の形で抽出し解析する手法や、逆に、半語・半言語で記述した仕様を満足する C/E ネットを設計するための手法が提案されている。半語・半言語で C/E ネットの解析や設計を行う利点は、並行動作を正しく、しかも自然な形で簡潔に表現できるということである。以上のように、半語・半言語は、C/E ネットの解析のための手法であり、それ単独で用いられる手法ではない。

以上の手法は、何れも、解析や検証の手法としては有用だが、それ単独では、対象の概観や動作の理解がしにくいものである。そのため、実用的な離散型並列生産システムのモデルとしては利用しにくい。一方、以下に示す各手法は、いずれも、状態遷移グラフがベースの図的表現を持つため、対象のイメージや動作の理解がし易い手法である。更に、グラフ理論や線形代数を用いた振る舞いの静的・動的な解析、シミュレーションベースでの動作チェックやモデル検査による動作検証など、解析や検証の方法も、個々の手法に合わせて各々提案されている。

ペトリネット

ペトリネットは[11)]、理論的計算モデルであり、「並列動作」，「分散状態」，「資源の概念」を表現可能なオートマトンである。グラフ的構造の図的表現と数学的モデルの両方を持ち、グラフ理論を利用

した静的な解析と線形代数による動的な解析(どのような振る舞いをするかを調べる解析)の双方が可能である。また、有界なペトリネットに関しては、モデル検査による論理検証も可能である。

通常のペトリネットをベースに、以下示すような、様々な種類のペトリネットが提案されている。

・時間ペトリネット：　通常のペトリネットに確定的な時間の概念を導入したものである。各トランジションに対し、発火可能になってから実際に発火するまでの遅延時間を付随させる“発火遅延モデル”、各トランジションに対し、発火継続時間を付随させる“発火継続モデル”、各プレースに対し、それが発火に利用できるようになるまでの遅延時間を付随させる“プレース時間モデル”等がある。

・タイムペトリネット：　各トランジションに対し、発火可能になってから実際に発火するまでの遅延時間の下限および上限を付随させたモデルである。

・確率ペトリネット：　時間ペトリネットで導入された時間を確率分布の形で置き換えたモデルである。

・ファジィペトリネット：　不確実性を持つ対象に対する推論を行うルールベース・システムの表現に用いられる。

・ハイブリッドペトリネット：　連続的状態変化を表現した部分(“連続ペトリネット”)と離散値を持つ通常のペトリネットを融合させたモデルである。ここで、“連続ペトリネット”とは、連続値を持つプレースとその値を連続的に変化させるトランジションを持つものである。

・高水準ペトリネット：　大規模なシステムをモデル化するとき、通常のペトリネットでは同型な構造を持つ部分ネットが数多く生成され、システムを記述する上で大きな負担となる。この問題を解消するために提案されたのが“述語/トランジションネット”　や“カラーペトリネット”などがある。

・オブジェクトペトリネット　トークンがオブジェクト、すなわち、データとそれを操作するためのメソッドをあわせたものとなっている。

有限オートマトン

有限オートマトン（finite automaton、FA)または有限状態機械（finite state machine、FSM）とは、有限個の状態と遷移と動作の組み合わせからなる数学的に抽象化された「ふるまいのモデル」である。デジタル回路やプログラムの設計で使われることがあり、ある一連の状態をとったとき、どのように論理が流れるかを調べることができる。有限個の「状態」のうち 1 つの状態をとる。ある時点では 1 つの状態しかとらず、それをその時点の「現在状態」と呼ぶ。何らかのイベントや条件によってある状態から別の状態へと移行し、それを「遷移」と呼ぶ。それぞれの現在状態から遷移しうる状態と、遷移のきっかけとなる条件を列挙することで定義される。

有限オートマトンは様々な問題に応用でき、半導体設計の自動化、通信プロトコル設計、構文解析などの工学面での応用がある。生物学や人工知能研究では状態機械（群）を使って神経系をモデル化し、言語学では自然言語の文法をモデル化したりする。

時間オートマトン

時間オートマトンは [11),12)]、有限オートマトンの拡張であり、システムを離散的なイベントと連続的な時間経過の両方による状態遷移で記述するモデルである。時間オートマトンでは連続的な時間経過を扱うため、一般にクロックによる同期のない物理的な制御対象との相互作用を自然な形で記述可能である。また、離散イベント間の時間制約を記述可能であり、時間的な挙動と離散的な状態遷移が相互に影響しあう動作を記述できる。さらに、イベントと時間のみに着目し、他の観測量、制御量や具体的な入出力データなどを捨象するため、効率の良いモデル検査が可能で、実用に耐える UPPAAL のようなツールも近年整備されてきている(2.5 章参照)。

時間オートマトンの例を Fig.2.5 に示す．図中の四角はロケーションを表し，内部の s0，s1，s2 はロケーション名である．初期ロケーションは s0 で，各ロケーションに付随するクロック制約はロケーション不変条件である．遷移の枝(矢印)に付随するラベルは，それぞれアクション名，ガード条件，リセット集合である。なお、時間オートマトンは、リアルタイムシステム(注 2.1)や論理回路の仕様記述と検証のために、ω-オートマトン(注 2.2)を拡張し、開発されたものである。

(注 2.1)ジョブの実行が命令された時、その処理を設定された時間通りに実行することが求められる制御工学における概念の一つである。

(注 2.2) ω オートマトンとは古典的な有限オートマトンの受理の概念を、無限長の語に拡張したオートマトンである。無限語の受理の定義の流儀にはいくつかあるが、もっとも単純なものが「指定した受理状態のいずれかを無限回訪れるような入力語を受理する」という条件であり、Buchi 受理条件と呼ばれる。

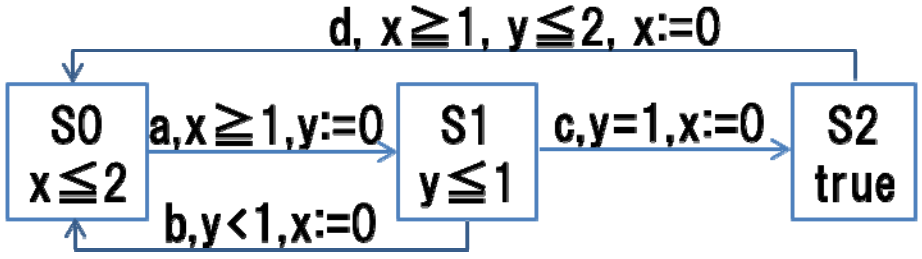


Fig.2.5: 時間オートマトンの例-

ステートチャート

リアルタイムシステムを記述するためには、オートマトンが広く使用されていたが、状態数が非常に多い時は、オートマトンをダイヤグラム的に記述することが困難である。この問題に対処するために開発されたステートチャートは、有限オートマトンを拡張し、状態(ロケーション)を繰り返し分割しながら、AND/OR 関係にあるサブ状態の組に展開できるようにし、仕様記述のモジュール化や、階層化及び平行化を実現可能にしたものである[13)]。

ステートチャートの主要な特徴は以下の通りである。

・階層化（OR 分割）： 階層化は状態(ロケーション)をクラスタ化し、状態と状態集合を閉じた等高線により表現する。状態の等高線の境界で作られる遷移は、全てのロケーションの集合に適用される。Fig.2.6(a)で示すように、イベントｂは状態 A または状態 C から状態 B に状態遷移させる。階層化は(b)のようにＡとＣをＤに抽象化して､Ｄの境界から２つのイベントｂを１つのイベントｂで置換する。Ａと Ｃの結合 D は排他的 OR であり、D に入ると、A またはＣに遷移する。

・並行化（AND 分割）： AND 分割されるロケーションは AND 構成要素のサブロケーションのすべてからなり、AND 分割はサブロケーションのカルテジアン積として表現される。Fig.2.7(a)の図は(b)と等価であり、(b)のＳとＴの両方が並行動作する。ロケーション集合を｛S,T｝で表現すると、初期状態は｛V,W｝である。この状態からイベントｅが発火可能となると｛X,Y｝に遷移して、イベントｋが発火可能となると｛V,Z｝に遷移する。このような AND 形式の分解により直交状態ができて、ロケーション数の膨張が解決でき、独立した並行な状態が記述できる。

・ブロードキャスト通信：上記の“並行性”による直交状態の全ての遷移は、ブロードキャスト通信により実現される。これは、外部のイベントの生起は全ての直交状態の遷移を引き起こすということである。

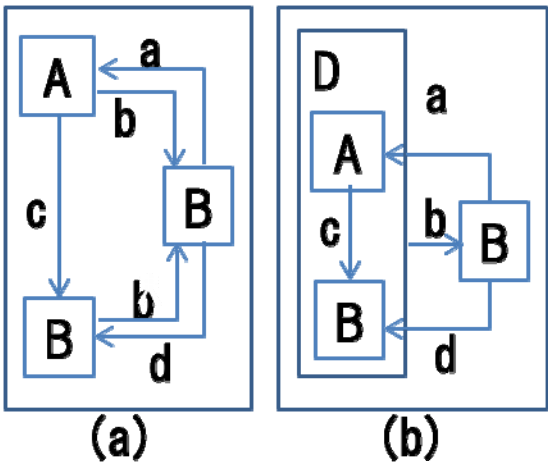


Fig.2.6: ステートチャート（階層化）

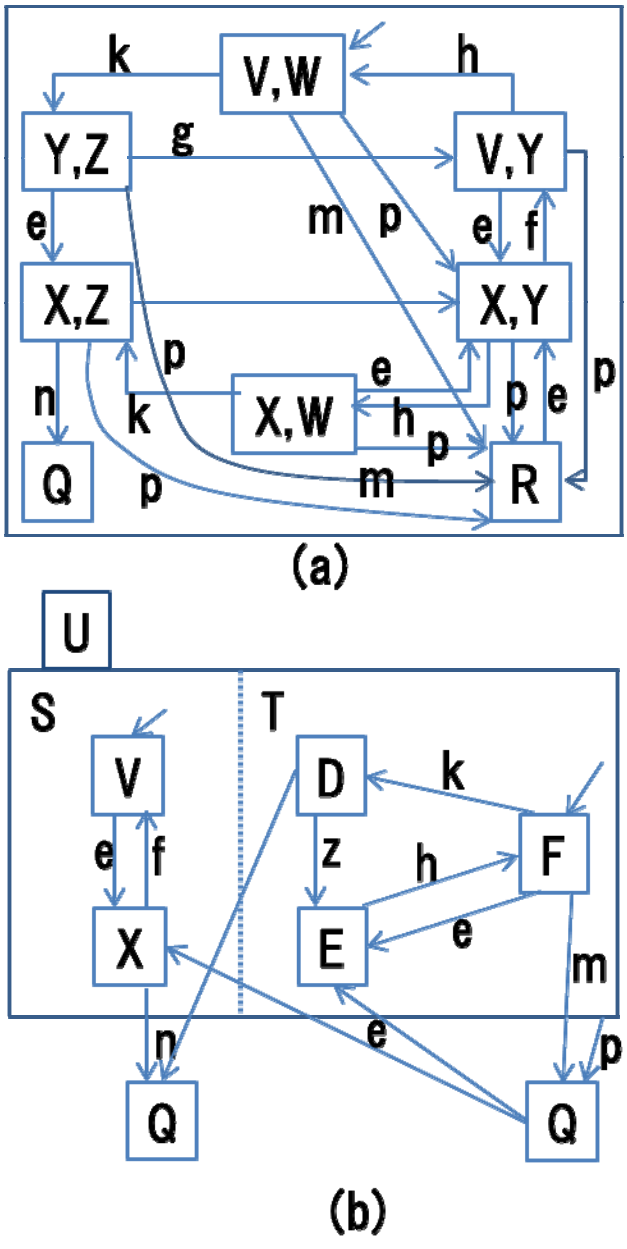


Fig.2.7: ステートチャート（並行化）

時間ステートチャート

時間ステートチャートは[1),2)]、ロケーションの組み合わせ爆発への克服が可能なステートチャートを、タイミング制約の記述が可能な時間オートマトンにより形式化したものである。以下、時間ステートチャートの図的表現、意味、動作について示す。

（1）図的表現

Fig.2.8 は時間ステートチャートの図的表現である。A～F はロケーション、x はクロック変数、a～d は入力アルファベット(イベント)、x:=0 はクロック変数のリセット式、x＜2 などはタイミング制約である。ロケーションの X と Y の間の破線の境界線は X と Y が AND 分割されていることを表している。また、破線で仕切られていない D と E は、C の OR 分割であることを表している。ロケーション X では、初期状態が A であり、a が入力される(発火可能となる)とクロック x がリセットされてロケーション B に遷移し、ロケーション B で b が入力されて(発火可能となって)タイミング制約式 x＜２が満たされると A に遷移する。ロケーション Y も同様であり、X と Y の動作は同時並行的に進んでいく。

例えば、Fig.2.8 の時間ステートチャートへ、(a,2)→(b,3.5)→(a,4) （ただし、(a,2)は、「イベント a が時間 2 で生起する(発火する)」を意味し、→は、イベントの生起(発火)の順序関係を表す。） が入力されると、ロケーション X, Y の動作は各々次のようになる。

	(a,2)		(b,3.5)		(a,4)	
X： ＜A,[0]＞	→	＜B,[0]＞	→	＜A,[1.5]＞	→	＜B,[0]＞
	(a,2)		(b,3.5)		(a,4)	
Y： ＜F,[0]＞	→	＜D,[2]＞	→	＜F,[0]＞	→	＜D,[0.5]＞

ただし、[]内の値はクロック x の値を表す。

（2）意味

時間ステートチャートの意味は、上の例で示すような、イベント σ_i と時間 τ_i の順序対 (σ_i, τ_i) の無限の遷移列 $(\sigma_1, \tau_1) \rightarrow (\sigma_2, \tau_2) \rightarrow (\sigma_3, \tau_3) \rightarrow$ として表現される。ここで、τ_i の領域は実数であり、かつ、以下の条件を満たす。

・初期条件： $\tau_0=0$ が開始時間である。

・単調性： $\forall i \geqq 0,\ \tau_i < \tau_{i+1}$

・前進性： $\forall t \in R,\ \exists i,\ \tau_i > t$

（3）動作

時間ステートチャートの動作は、以上のような「意味」の定義を使って得られるもの（即ち、振る舞い）である。

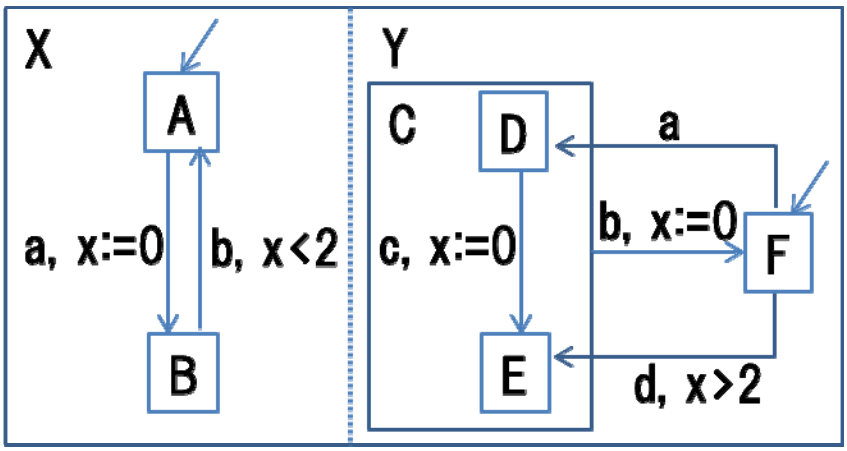


Fig.2.8:　時間ステートチャートの例

Function Block

Function Block(FB)は[14)~17)]、プログラマブルロジックコントローラ（PLC）専用のグラフィカルなプログラム言語である。「入力パラメータ」と「出力パラメータ」をもち、複数の機能を組み合わせた制御（処理）が部品化され、1つの命令のように簡素化された“FB”をノードとし、FB同士を接線で繋ぐネットワーク構造としてモデリングされる。従って、複雑な動作が直感的理解しやすい形でモデル化できる。Function Blockは、PLC使用の制御系の設計のみをターゲットとしたモデルである。そのため、離散型並列生産システムの例として先に挙げた対象のうち、FAシステムには利用できるが、バッチプラントやネットワーク型生産システムへの適用は難しい。

実時間並行ソフトウェアシステムと離散型並列生産システムの振る舞いの違いについて

先に述べたとおり、後半で紹介した状態遷移グラフがベースのオートマトンやステートチャートがベースとなるモデルは、その図的表現により対象のイメージや動作の理解がし易い上、解析や検証の方法も充実しているものが多い。また、いずれの手法も、離散型並列生産システムのモデル化や検証にそのまま利用できるものにはなっていないが、利用できる性質を部分的には備えている。特に、時間ステートチャートは、並行・階層性の表現、人が見て理解し易い、リアクティブ性や実時間制約を扱えるなど、離散型並列生産システムの挙動表現モデルが満たすべき要件を最も多く満たしている既存手法である。なお、時間ステートチャートは、航空機、自動車、携帯電話、情報家電などのソフトウェアシステム(以降、“実時間並行ソフトウェア”と記す）の振る舞いを記述し検証することを主目的とするモデルとなっている。

以上を踏まえ、本稿では、離散型並列生産システムのベースモデルとして時間ステートチャートを採用した。そして、実時間並行ソフトウェアと離散型並列生産システムの振る舞いとの間の違いを考慮しながら、生産システムの振る舞いを記述する上で必要な要件を満たすようにするための拡張を時間ステートチャートに対して行った[38)~41)]。

Fig. 2.9 は、実時間並行ソフトウェアと離散型並列生産システムの振る舞いとの間の本質的な違いを表す図である。両者の大きな違いは、実時間並行ソフトウェアは、通常、あるジョブの実行が終わるまで次のジョブの実行は行わないという実行形態が取られるシステムであるのに対し、離散型並列生産システムとは、有る期間内に発生した複数ジョブを、出来る限り効率の良い装置割り当てや装置使用順序の下で、同時並行的に処理するという実行形態が取られるシステムであるという点である。以下、具体例で説明する。

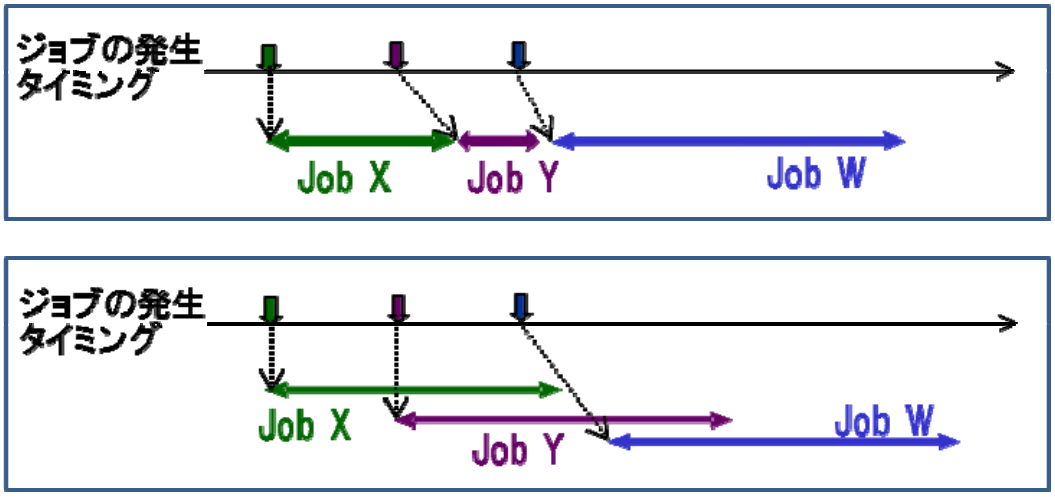


Fig.2.9:　実時間並行ソフトウェア(上図)と離散型並列生産システムの振る舞い(下図)の違い

実時間並行ソフトウェアの例として、自動車制御システムについて述べる。Fig. 2. 10 は自動車制御システムの仕様を時間ステートチャートで記述したものである[1)]。本システムは、ドライバや外部環境とインタフェースして、アクセル、エンジン、トランスミッション、ブレーキから構成される。アクセル、ブレーキは、ドライバからの直接的な命令（つまり、アクセル操作やブレーキ操作）により作動し、エンジンやトランスミッションはドライバのエンジン操作、アクセル操作により発生する間接的な命令で作動する。このようなシステムにおいて、例えば、[アクセル，ブレーキ，ブレーキ，アクセル，･･･]のように、複数の命令(ジョブ)が順番に発生する場合、システムは、通常、その発生順にジョブを実行し、一つのジョブの実行が終了するまでは次の実行には移らない、という振る舞いをする。また、アクセル

に対する命令(ジョブ)はアクセルとエンジンの協調動作で遂行され、ブレーキに対する命令はブレーキとトランスミッションの協調動作で遂行されることが、Fig.2.10 より分かる。つまり、実時間並行ソフトウェアにおける“並列動作”とは、一つのジョブを遂行するために必要な複数のプロセス同士の協調動作であり、複数のジョブが同じシステム内で同時並行的に実行されることはない。

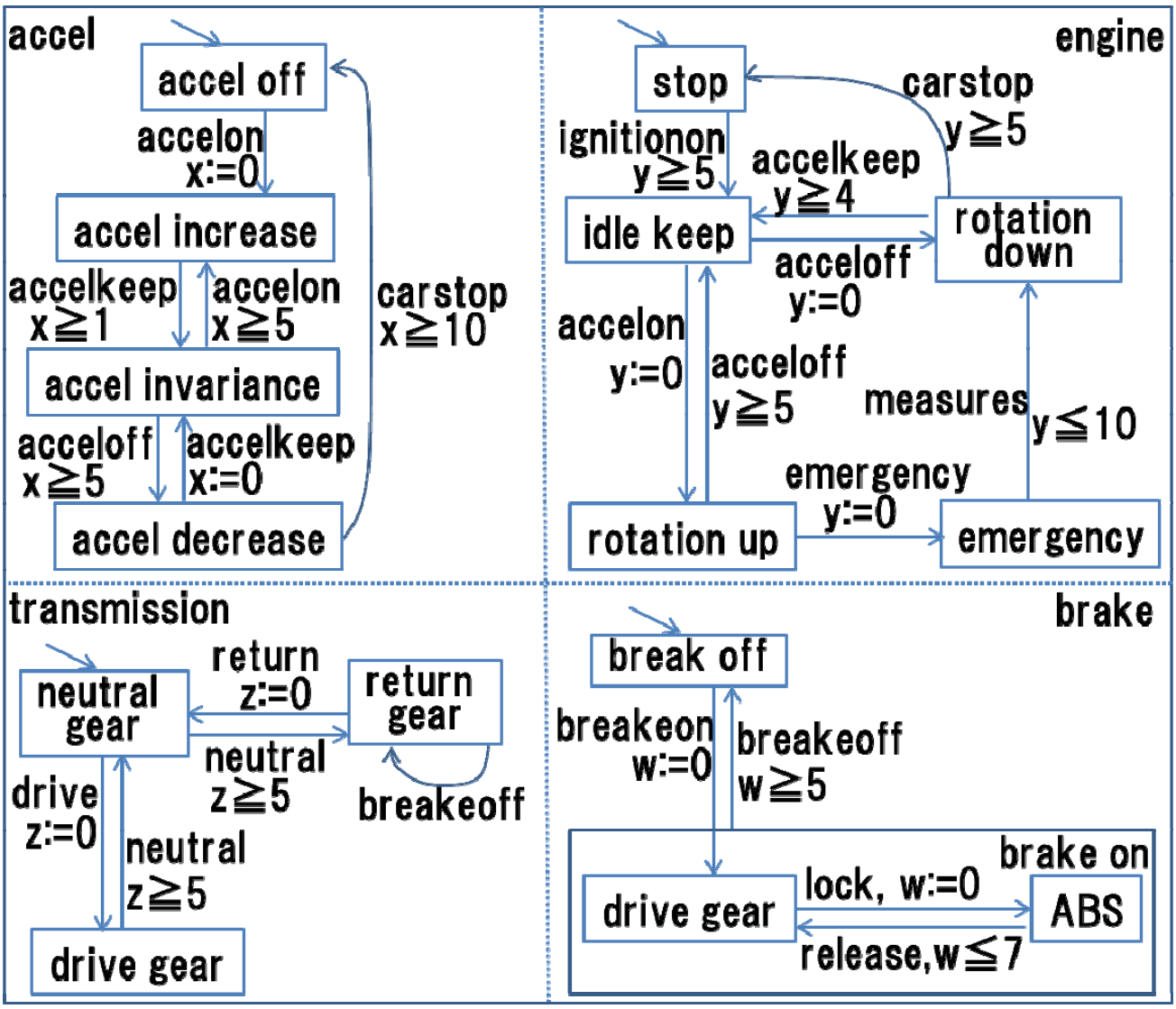


Fig.2.10:　自動車制御システムの仕様を時間ステートチャートで記述した例

一方、離散型並列生産システムの場合、複数の命令(ジョブ)に対し、その発生順にジョブを実行することはせず、それら一まとまりのジョブに対し、通常、総所要時間を出来るだけ小さくするような生産スケジューリングを行い、その結果得られたスケジュール(ガントチャート)が示す装置使用順序に従ってシステムは実行することとなる。つまり、予めスケジュールされた装置使用順序に従い、複数ジョブが同時並行的に処理されていくということである。Fig.2.11 のガントチャートは、装置１～３を持つ生産システムで、製品 X,Y,Z を各一つずつ作る際の生産スケジュールの例を表している。ここで、各製品 X,Y,Z の製造工程は、順に、(X1→X2→X3), (Y1→Y2, Y2'→Y)、(Z1→Z2) である。

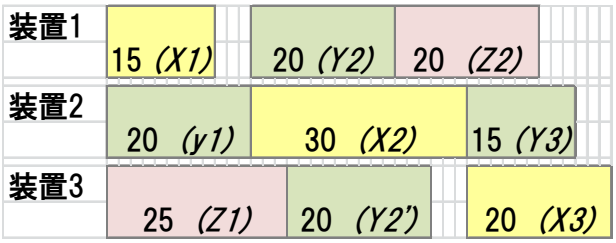


Fig.2.11:　離散型並列生産システムのガントチャートの例

2.4 制御系設計手法について

近年の離散型並列生産システムは、巨大化･複雑化が進んでいる。そのため、開発に携わる人間の数が多く、システムへの要求変更も多い、といった特徴がある。このような背景から、(Ⅰ)開発者の正確な対象認識や、開発者同士あるいは開発者･ユーザ間での確実な情報伝達に適したシステム動作記述モデルが求められている。また、顧客ニーズの多様化により、多品種少量や変種変量生産が求められていることはよく知られている。このような離散型生産システムは、事前のスケジューリングにより作成される計画に基づき運転操作される。このとき、一般に、処理所要時間には不確定性があるため，時間駆動で表現されたスケジューリング結果を事象駆動の制御操作に変換して実装することが不可欠である。そのため、(Ⅱ)時間駆動の計画系から事象駆動の制御系への変換を系統的に行い、人手を介さずに制御系の作成や変更を行えるような制御系設計のためのフレームワークも求められている。

(Ⅰ)について

冒頭で述べたとおり、本研究では、まず離散型並列生産システムの動作記述モデルを開発し、次に、そのモデルに基づく制御系設計手法を開発することを目的の一つとしている。なお、この種のシステム

の既存のモデリング手法に関する現状、及び、それを踏まえて本稿で提案するモデルの開発方針については、上で述べた通りである。

(Ⅱ)について

計画系から制御系への変換を系統的に行う方法論に関する現状は次の通りである。まず、計画系での問題解決(スケジューリング)に関しては、混合整数計画法(MILP)や動的計画法(DP)のような OR 的手法やその他の数理計画法を用いる手法が用いられる。また、そのような数理的扱いが困難な複雑な構造を持つ場合に対しては、遺伝的アルゴリズムやシミュレーテッド・アニーリング法などのヒューリスティクス的手法が様々に提案され、用いられている[44)~47)]。

・混合整数計画法(MILP)： 決定変数が連続変数及び離散変数で、制約条件や目的関数が全て線形の式で表現された数理計画問題を混合整数線形計画問題と呼ぶ。

・動的計画法(DP)： 動的計画法は対象となる最適化問題を複数の部分問題に分割し、求められている以上の最適解が求められないような部分問題を切り捨てながら解いていく手法を動的計画法と呼ぶ。状態数があまり大きな問題だと扱えない。

・遺伝的アルゴリズム： 自然界における交叉・突然変異・淘汰・適者生存といった進化の仕組みをモチーフに、最適化問題の解を求めるアルゴリズムである。ヒューリスティクス的方法であるため、必ずしも厳密解が得られるわけではないが、比較的短時間で厳密解に近い解を得ることはできる。

・シミュレーテッド・アニーリング(SA 法)： 大域的最適化問題への汎用の確率的メタアルゴリズムである。広大な探索空間内の与えられた関数の大域的最適解に対して、よい近似を与える。山登り法と違い、簡素な実装で安定して局所解からの脱出が可能であるが、計算量が多い、パラメータに性能が依存、という問題がある。

しかし、上記方法のいずれの場合も、その解は時間駆動の操作の列として与えられ、それを事象駆動の制御操作に変換して実装するための方法論が必要である。

また、実行系での制御の実装では、ペトリネットを使った方法[48)]、マトリックスベースの手法[49),50)]、FB(Function Block)[14)~17)]を使った方法、或いは、バッチ型化学プラントでの国際標準規格 S88[22)]を使った方法[22)]等、様々な実装方法や開発支援環境が提案されている。

・ペトリネットを使った方法： 文献 48)で提案される“K-NET”では、制御対象の使用を入力することにより、対象の振る舞いを表すペトリネットが自動生成され、シミュレーションベースでの動作チェ

ックができる。ただし、本方法では、制御対象に関する雛形モデルが何らかの形で事前に準備されている必要がある。

・マトリックスベースの手法：　文献 49),50)で提案される MDEC では、ペトリネットで表現されたシステムを対象に、Petri net 上に離散事象システムの状態方程式を持たせ、この状態方程式を従来の マーキング遷移方程式（MTE)と併せて使うことによって、完全な動作記述ができるようになっている。従って、MDEC では、実行時のシステム内部の状態変化をコンパクトに表現でき、次の操作（発火させるべきトランジション）をマトリクス計算で求めることが出来る。更に、MDEC の拡張である MDEC2 [34] では、イベントの生成や発火可能タイミングの時間制約などを扱うことができ、MDEC では扱えないより複雑な対象システムの振る舞いも従来法の枠組みの中で簡潔に扱える。

・FB(Function Block)を使った方法 [14)~17)]：　工場などの自動機械の制御、エレベーター、自動ドアなど、身近な機械の制御装置として良く知られた PLC（Programmable Logic Controller）は制御装置であり、中身はコンピュータそのものである。したがって，システム全体として安全認証を行うときはハードウェアだけでは不十分で，ソフトウェアに対しても何らかの判断基準が必要になってきている。FB は、PLC で使用されるソフトウェアのモデル化と動作検証を行うための手法である。

しかし、上記いずれの方法においても、計画系から実行系への動的かつ円滑な情報伝達の問題に関しては、系統的な解決法が与えられているとは言えない。以下では、まず、この種の問題に対する近年の取り組みについて示す。また、製造現場においてこの問題に対処するための汎用的な方法が得られにくい理由について示す。

上記問題に対し、近年、業務・計画系と制御系を円滑に「つなぐ」べき情報システムとして、MES（Manufacturing Execution System）という概念が提唱されており、システムの標準化や部品化、あるいは各サブシステムの統合等に関する観点からの議論や試みがさかんに行われている。MES が生まれてきた背景には、製造機械のインテリジェント化がある。ディスクリート産業では、機械加工の自働化といえば古くからＮＣ工作機械などがあった。それ以外の加工装置類もシーケンサーの発達のおかげで、柔軟な制御ロジックならびに通信インタフェースを持ちうるようになってきた。そこで、それまでは点状に孤立した機械群から成り立っていた工場のショップフロアを、協調し制御できる技術基盤ができてきたのである。また、プロセス産業では'80 年代後半から DCS(Distributed Control System)による分散制御が当たり前となっていた。また、加工装置中心のライン生産とは異なる、人間による組立生産工

程においても、中間部品やワークをバーコード／RFID でリアルタイムに追いかけていく、POP（Point of Production）と呼ばれるシステムが広まってきた。さらに、上位からきた生産指示情報（最終製品レベルでの生産オーダー）をショップへの製造指示情報（部品・材料レベル＋工程レベルでの製造指図）にかみ砕くための仕組みである工場スケジューラも普及してきた。こうした流れが一つに合わさって、MES というシステム群が生まれた。ただし、ディスクリート型生産とプロセス型生産の間には、非常に大きなギャップがある。更に、製造現場というものは総じて個別性が強い。こうした個別性の壁があるため、ＭＥＳは、どの産業、引いてはどの製造現場にも適用できるような汎用的方法論としてシステム化しにくいという特性がある。そのため、本研究で扱っている生産ラインレベルでは、現在もなお、各企業が独自に蓄積してきたノウハウに基づき、適宜、差立や、制御プログラムの作成・追加・変更など、かなりの手作業を介してプラント運転しているのが現状である[18)~21)]。

以上を踏まえ、本研究では、離散型並列生産システムのための汎用的な制御系設計のためのフレームワークを提案する。本提案方法は、離散型並列生産システムのための挙動表現モデルとして本稿で開発する拡張時間ステートチャート(ETSC)をベースモデルとすることにより、制御情報の抽出が機械的に行えるようになるため、モデルベースで判りやすい方法となる(詳細は 4.2.2)。また、本方法の適用可能範囲としては、少なくとも、本章の冒頭で示した 3 種類の生産システム（ネットワーク型、組み立て式、バッチ式）には何れも適用できる程度の汎用性は持たせる。このように、ある程度の汎用性を、上述の「個別性」に関わらず持たせるための仕組みは、挙動表現モデル ETSC から制御情報を抽出するために導入する 5 種類の“テンプレート”と、各テンプレートの内部構造によって与えられる。ここで導入したテンプレートとは、制御に必要な情報を、スケジュールによって動的に変わる部分と、スケジュールによって変わらない静的な部分とに分けて記述できるような、5 種類の表形式のテンプレートのことである。また、テンプレートの内部構造とは、取り出すべき情報の種類の指定と、取り出された情報の記述形式で決まるフォーマットのことである。本研究では、これらのテンプレートに埋め込むべき情報を、ETSC からシステマティックな手順で抽出するための方法を合わせて示す。具体的には、導入する 5 種類のテンプレートの間には階層関係があり、その階層関係を利用することにより系統的かつ機械的な手順で抽出出来る。更に、このフレームワークの下では当然考えておかねばならない、実行時間の遅延(もしくは“ずれ”)が許容範囲を超えた場合の対処方法、具体的には、ずれの大きさが許容範囲を超えたときに行われる再スケジュールリングの結果を、現在実行中の当初スケジュールに動的に繋ぐ方法を示す[28),29)]。

2.5 動作検証手法について

離散型並列生産システムでは、処理時間の不確定性や操作上の非決定性によって生じる(生産計画からの)ずれにシステムが何処まで耐えられるかに関する動作検証が必要である。非同期分散システムの動作を比較的厳密にチェックするための方法としては、モデルと検査項目を入力として与え、ラベリングアルゴリズムを使って検証を行うモデルチェック型の検証法が一般的である。そのためのツールも多数開発されており、SPIN、UPPAAL、NuSMV などが良く知られている。

・SPIN[51),52)]： SPIN は並列離散システムのためのモデリング、シミュレーション、及び検証のための統合ツール環境である。システムの振る舞いは、そこで使用されるモデリング言語 Promela で記述され、並行動作する有限オートマトン(以降、“プロセス”)間のチャネルを介したメッセージ通信は、同期通信（1 対 1 通信）、非同期通信（キューによるバッファリング）の両方で記述する事ができる。

・UPPAAL[53)～56)]： UPPAAL はリアルタイムシステムのモデリング、シミュレーション、及び検証のための統合ツール環境である。UPPAAL ではシステムモデルをプロセス毎に時間オートマトンでモデルを記述させる。そのため、通信を伴う並行処理、排他処理を明確に記述でき、時間を明示的に扱うことが可能であり、GUI ベースでモデルを作成可能である。通信様式は同期通信（1 対 1 通信）である。また、UPPAAL では、システムが満たすべき性質を計算木論理 CTL(Computation Tree Logic)による検証式で記述させ、モデルチェッキングアルゴリズムにより検証される。

・NuSMV[57),58)]： NuSMV はシンボリックな手法を用いた最初のモデル検査ツールである SMV(Symbolic Model Verifier)から派生したツールのひとつである。対象システムの動作は SMV 言語で記述される。SMV 言語ではシステムの状態を変数の組み合わせで表現させ、共有メモリ通信により、同期、非同期の有限状態システムでも記述することができるように設計されている。検証されるべき性質は計算木論理 CTL(Computational Tree Logic)と線形時相論理 LTL(Linear Temporal Logic)で表すことができ、モデルチェッキングアルゴリズムにより検証される。

離散型並列生産システムの場合、これまで、その振る舞いを的確に表現できるような肝心の「モデル」が存在しなかったため、モデル検査による検証が行えなかった。そのため、これまでは、テストデータを使ったシミュレーション程度のチェックしか行われてこなかったという現状がある[22)]。この点を踏まえ、本論文では、本稿で提案する挙動表現モデルを使用し、離散型並列生産システムの適応限界を調べるためのモデル検査型の検証方法を提案する。なお、この種のシステムの適応限界を調べるためのある

程度厳密な別の方法としては、モデル検査型の方法以外にも、確率的 PERT(Stochastic PERT)でクリティカルパスになる可能性のあるパスに着目する方法や、計算機代数の理論である QE（Quantifier Elimination）を使用し制御仕様の集合が定める実行可能領域に着目してそれを求める方法等もある[59)~65)]。

・確率的 PERT を使った遅延リスク評価方法[59)~62)]：

PERT のクリティカルパスに着目した生産システムの動作解析手法では、通常、生産システムの振る舞いを PERT ネットワークを用いて表し、それを用いて解析する方法が標準的な手法として提案されている。さらに、その工程の所要時間が確率的に変動する場合についても、様々に研究がすすめられ、クリティカルパスの総所要時間の確率密度関数の評価や近似などの研究が進められている。

クリティカルパスの総所要時間の遅延リスク評価としては、例えば、遅延リスクがある割合以下になる総所要時間の上限を求めたり、或いは、ある総所要時間が定められた納期以下になる確率を求めることができる[42)]。

・QE（Quantifier Elimination）を使った遅延リスク評価方法：

QE とは、限量記号の入った論理式を、それを等価で限量記号を持たない式に変換する計算方法のことである。QE のアルゴリズムでは、与えられた多項式系に対し，変数空間を各多項式の符号が不変である領域に分割する代数的方法「CAD」を使う。QE の実行では、例えば、まず、限量記号つきの不等式制約の集まりを入力として与える。このとき、QE は、各不等式を、各々、それ自体の真偽を表す論理式と見立て、それらをそれと等価で限量記号の無い式に変換し、結果として、自由変数(パラメータ)の取り得る領域を出力する。

QE を使って処理時間の不確定変動に対する離散型並列生産システムの適応限界を調べる方法では、まず、対象システムの制御仕様と要求仕様を QE への入力形式である不等式制約で記述する。この時、各処理の遅延の長さは、自由変数、つまり、パラメータで表す。以上を入力データとして QE を実行すると、パラメータ全体が取り得る領域、つまり、実行可能領域が制約条件式の集まりとして得られる[41),42)]。

納期の遅延リスク評価は、上述のようにして得られる制約条件式全体が成り立たない確率を、モンテカルロ法的にチェックすることにより可能である。具体的には、各パラメータの値の発生が指数分布に従うとき、パラメータの取る値の組が実行可能領域内に収まらない確率を求めると遅延リスクが得られる[41)]。

第 3 章　離散型並列生産システムの挙動表現モデルの提案

2 章で述べたとおり、並列分散型システムのための既存のモデリング手法は、いずれも、離散型並列生産システムのモデル化や検証にそのまま利用できるものにはなっていない。しかしながら、実時間並行ソフトウェアと離散型並列生産システムの振る舞いの挙動表現モデルとしては、並行・階層性、視覚的理解のし易さ、リアクティブ性や実時間制約を扱えることなど、共通に求められる要件も多い。時間ステートチャートは、既存手法の中で、このような共通に求められる要件を最もたくさん満たしているモデルである。従って、本章では、離散型並列生産システムのベースモデルとして時間ステートチャートを採用し、生産システムの振る舞いを記述する上で必要な要件を満たせるようにするための拡張を行うことにより、離散型並列生産システムの挙動表現モデルの導入を行う。具体的には、まず、離散型並列生産システムの動作的特徴を洗い出し、その中で、時間ステートチャート(TSC)では扱えない性質をピックアップする。次に、それらの特徴を扱えるようにするために TSC に対して行う拡張の内容について示す。更に、得られた拡張型の TSC モデル（「拡張時間ステートチャート」（ETSC））の形式的な定義と、その動作の意味定義を与える。

3.1　離散型並列生産システムの特徴的動作

離散型並列生産システムの動作の特徴は以下の通りである。

① システム全体が、遂次動作と並列動作が混在かつ階層構造を持つ中で事象駆動の動作をする。

② いくつかの動作には時間の制約がある。

③ いくつかの処理の進行には非決定性がある。

④ いくつかの処理の所要時間には不確定性がある。即ち、ある確率分布に従って連続的かつ不確定に変動する。

⑤ 許容される処理所要時間には上限があり、実行時の遅れを調整するために取れる遅延時間には限界がある。(生産システムの場合、個々のイベントの発火可能期間(特に上限)が本質的に重要な制約条件となる。)

⑥ 一つのジョブの実行は複数のプロセスの協調動作により実現される。(複数プロセスの並列処理)

⑦ 複数の内部イベントと複数の外部イベントが混在する。(ただし、外部イベントとは、ジョブの発生や生産要求の変更等、外部から与えられるイベントである。また、内部イベントとは、外部イベントへの対処の過程でシステム内で生成されるイベントである。具体的には、複数ジョブが同一装置を使用する際の競合解消のためのイベントや、一つのジョブのある製造工程において使用

される装置と次に使用される装置との間で受け渡される協調のためのイベント等がある。)

⑧ 一つの生産システム内で複数のジョブが同時並行的に処理される。

2 章で述べたように、離散型並列生産システムの挙動表現モデルが満たすべき要件を部分的に満足する既存手法として、ⅰ)プロセス代数、ⅱ)時相論理、ⅲ)半語・半言語、ⅳ)ペトリネット、ⅴ)時間オートマトン、ⅵ)ステートチャート、ⅶ)時間ステートチャート(TSC)、などがある[1)~17)]。Table3.1 は、上記手法ⅰ)～ⅶ)が、各々、上記①～⑧の実現や扱いを可能とする上で必要な表現や仕組みを備えているか否かをまとめた表である。(表現や仕組みを備えていない、或いは不明な箇所は空欄になっている。)この表より、時間ステートチャートが、離散型並列生産システムの動作的特徴を表現するための表現や仕組みを他の手法よりも多く備えていることがわかる。また、時間ステートチャートで扱えない仕組みは⑤，⑦，⑧であることが分かる。従って、本稿では、⑤，⑦，⑧を扱えるように時間ステートチャートを拡張する。

Table 3.1: 離散型並列生産システムの特徴的動作は既存のモデル化手法でどの程度表現可能か

		プロセス代数	時相論理	半語・半言語	ペトリネット	時間オートマトン	ステートチャート	時間ステートチャート	拡張時間ステートチャート(提案手法)
①	階層性						○	○	○
	逐次動作と並列動作の混在	○		○	○	○	○	○	○
②	時間の制約条件		○		○	○		○	○
③	処理の進行の非決定性	○	○	○	○	○	○	○	○
④	処理時間の不確定性		○(不等式)		○(確率的表現)	○(不等式による表現)		○(不等式による表現)	○
⑤	イベントの発火可能期間の上限				○(タイムペトリネット	△(間接的表現)		△(間接的表現)	○
⑥	複数プロセスの協調動作を扱う仕組み				○	○	○	○	○
⑦	内部イベントの生成						○		○
	外部イベントと内部イベントの混在を複数同時に扱う仕組み						△	△	○ 3.2.1節
⑧	複数ジョブの並列実行を扱う仕組み								○ 3.2.2節

3.2 「時間ステートチャート」の拡張

上述したように、本研究では、離散型並列生産システムの挙動表現モデルを得るため、まず、既存手法の中で､離散型並列生産システムのモデル化に必要な要件を元々最も多く備えている TSC[1),2)]をベースモデルとして採用する。そして、それを、上述の「TSC では扱えない離散型並列生産システムの動作の特徴」(即ち、⑤，⑦，⑧)を扱えるように拡張し、「拡張時間ステートチャート」(ETSC)を得る。[38)~40),43)] 具体的には、次のような拡張を行う。

・離散型並列生産システムにおいて、本質的に重要な制約条件となる上記⑤の「個々のイベントの発火可能期間」を記述できるような記述文法を導入する。

・「内部イベントと外部イベントは分離処理可能であること」という TSC の持つ動作上の強い制約を排除するための動作機構(イベントプールに基づく機構)を導入する。また、当該動作機構には、各イベントに「発火可能期間」を付随させ、全てのイベント処理を、与えられた発火可能期間に基づいて処理（発火待ち，発火，抹消）させるようなシンプルな仕組みも併せて導入する。つまり、発火可能期間を持つ個々の生成イベントの制御を容易に行えるようにするために、モデルの実行セマンティクスを与える。（上記⑦の実現）

・各イベントに、どのジョブの処理に関わるイベントなのかが分かるようにするための“ジョブの識別子”(IDJ) を付随させる。そして、特定の識別子が付されたイベントの発生，発火の系列を辿ることにより、各ジョブの実行の流れを辿れるようにする。

なお、ジョブ識別子の導入による効果は、上記以外にも、同形な構造を持つ部分構造が数多く生成されるシステム記述上の煩わしさを解消できるということがある。具体的には、従来、ジョブごとに別々に与えていた振る舞いの記述を、同じ振る舞いをする(即ち同じ操作手順を持つ)ジョブの集まりにごとに一つずつ与えればよいこととなる。

3.2.1 イベントプール・マネージャに基づくイベント管理の仕組み

以下では、拡張内容について具体的に示す。

イベント記述文法

まず、準備として、ETSC の生成イベントの記述文法を定義する。この記述文法を使うと、生成イベントの発火可能期間を明示的に記述できる。また、複数のイベントを生成する場合は、それが全ての発火を要請する AND 型なのか、或いは、排他的に一つだけ発火することを要求する XOR 型なのかを指定できる。 以下、Fig.3.1 を使って本記述文法の諸定義を示す。

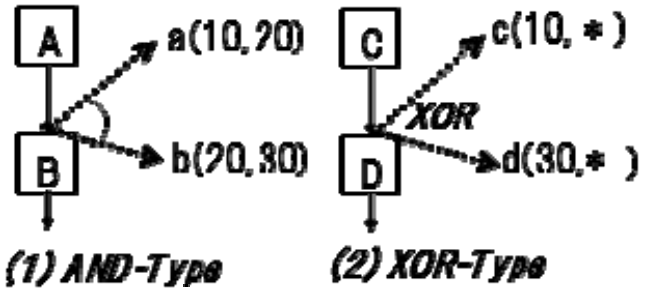


Fig.3.1: 各型のイベントの表現形式

・イベント生成タイミング：実線で示す遷移のアークの先端から伸びる破線アークがイベントの生成を表し、その先端の記号が生成イベントの名前(例えば、Fig.3.1 -“a”,“b”) を表す。つまり、「当該遷移(A→B)が起きた直後に、当該イベント(a と b)を生成せよ」となる。

・発火可能期間：生成イベント名に付随する整数値の組で表し、例えば、Fig.3.1-“a”の(10,20)) は、順に、「発火可能になるまでの期間」及び、「発火不能となるまでの期間」を表す。読み下すと、「イベント生成時点以降、10 クロックで発火可能となり、それ以降は、当該イベントが発火するか、或いは、発火不能となる時点(20 クロック後)までずっと発火可能であり続ける。」となる。

・イベントの型 : AND 型は指定された全てのイベントの発火を要請する。XOR 型は排他的に唯一つのイベントの発火を要請する。例えば、AND 型の関係を持つイベント a(10,20)と b(20,30)は、同じ実行パス上での双方の発火が要請されるが、XOR 型の関係を持つイベント c(10,*)と d(30,*)は同じ実行パス上ではどちらか片方のみの発火が要請される。ここで“*”は∞を表し、発火するまでずっと発火不能とならないことを意味する。

<u>イベントプール・マネージャ</u>

・イベントプール (：EP)：生成されたイベントの全てを一括して格納しておく場所の概念

・イベントプール・マネージャ(：EPM)：各生成イベントを指定された時間帯及び型に従って発火させるための管理を行う主体

EPM は、システムに導入する唯一つのグローバルクロックの時間を進めながら、各時点で、以下の操作を行う。

[EPM の基本操作]

(b1) システム上で生成されたイベントを EP へ登録する。

(b2)発火可能なタイミングとなった全てのイベントをシステム上へブロードキャストする。

(b3) 発火し終わったイベントを EP より抹消する。

(b4)発火可能期間満了後のイベントを EP より抹消する。

3.2.2 ジョブ識別子の挙動モデル上での振る舞い

ジョブ間、或いは、ジョブ-イベント間の“関係性”を保持する情報としてジョブの ID(ジョブ識別子(IDJ))を導入する。IDJ とは、直感的には製品タグに相当する。そのような IDJ は、ジョブの発生を引き起こす個々のイベント(即ち、“生産要求”を表すイベント)に、各々異なる IDJ を付す形で与える。そして、各 IDJ に対応付けられるジョブの実行は、当該 IDJ 付きイベントが、割り当てられた装置の振る舞いを表すオートマトン上で受理された時点で開始する。そして、当該ジョブが同じ装置上で処理されている間は、トークンの動きに IDJ を追随させる。そうすることにより、各時点において、「どのプロセスが実行中か」の認識に加え、「何を(どのジョブを)処理しているか」の認識もできるようになる。また、使用装置の切替時には、当該 IDJ を付したメッセージイベントをブロードキャストし、次に使用する装置(オートマトン)がそれを受理する。以下、ジョブの実行が終了するまで以上の振る舞いを繰り返し行わせる。以上のような IDJ の振る舞いを実現するための基本操作をまとめると、次のようになる。

[IDJ のための基本操作]

(c1) 準備：ジョブの実行開始を引き起こすイベント(即ち、生産要求発生を表すイベント) に、各々、異なる IDJ を与える。(Ex. reqX1, reqX2,…)

(c2) IDJ の受け渡し(受け渡し元) : イベント生成時には、常に、当該時点でトークンが持つ IDJ を当該生成イベントに付加し出力する。

(c3) IDJ の受け渡し(受け渡し先) : イベント受理時には、 ジョブ切り替えタイミング(具体的には、オートマトンの初期状態からの遷移を伴うイベント発火時)に限り、当該イベントに付された IDJ をトークンが受け取る。この時トークンが持っていた先の IDJ は破棄する。

(c3)の補足：ジョブ切り替え時以外のタイミングでのイベントの受理は、単にプロセス間の同期を取るためのものである。そのため、ジョブ切り替え(即ち、トークンの持つ IDJ の変更)は行わない。

3.2.3 拡張時間ステートチャートの例

[例 3.1] (拡張時間ステートチャートの図的表現の例)

Fig.3.2 は拡張時間ステートチャートの図的表現の例である。A,B,C,D,P,Q,W はロケーション、t はク

ロック変数、a,b,req はイベント、t:=0 はクロック変数のリセット式、5≦t≦50 はタイミング制約である。図の見方は次の通りである。ロケーション P と Q は、各々並行動作するプロセスを表している。ロケーションの P と Q の間の破線の境界線は P と Q が AND 分割されていることを表している。また、破線で仕切られていない A と B は P の OR 分割、C と D は Q の OR 分割であることを表している。ロケーション P では、初期状態が A であり、イベント req が入力される(発火可能となる)とロケーション B に遷移し、イベント a(20,*)と a(0,*)が XOR 関係で生成され、ロケーション B で b が入力される(発火可能となる)と クロック t がリセットされて A に遷移する。同様に、ロケーション Q では、初期状態が C であり、イベント a が入力される(発火可能となる)とロケーション D に遷移し、b(10,*)と c(10,*)が AND 関係で生成され、ロケーション D でタイミング制約 5≦t≦50 が満たされると C に遷移する。

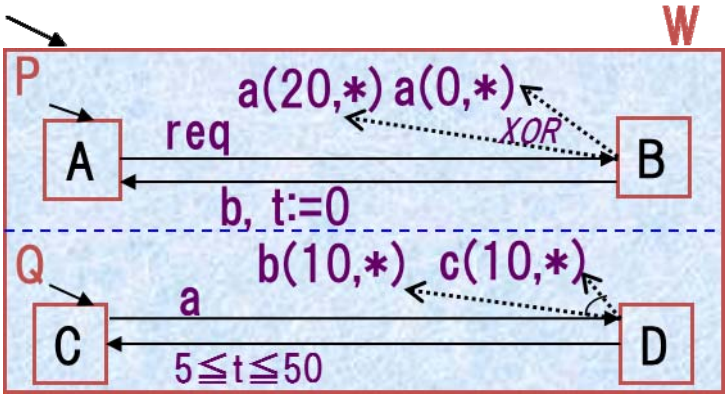


Fig.3.2: 拡張時間ステートチャートの例

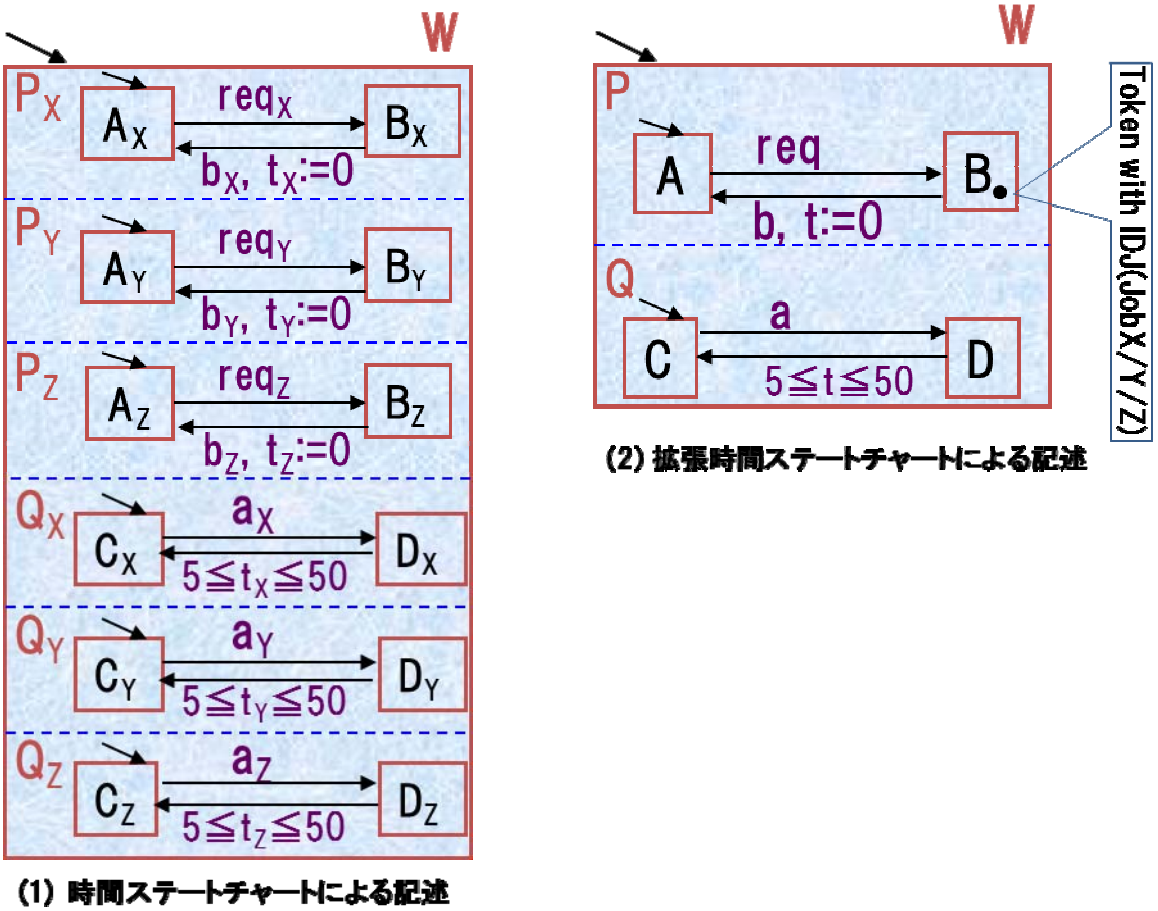
[例 3.2] (ジョブ識別子の効果)

Fig.3.3 は、ジョブ識別子による効果、即ち「同形な構造を持つ部分構造が数多く生成されるシステム記述上の煩わしさを解消できる」という効果を示す。Fig.3.3-(1)は、Fig.3.2 の ETSC を、ジョブの発生数をジョブ x，y，z の 3 つとするケースに限定し、IDJ を持たない通常の時間ステートチャートで記述したものである。(ただし、IDJ の効果を分かり易くするため、生成イベントの記述は省略してある。) Fig.3.3-(2)は、Fig.3.3-(1)と同じ振る舞いを、IDJ で拡張された ETSC で表したものである。(Fig.3.2 の ETSC から生成イベントの記述を省略したものと同じである。)

Fig.3.3-(1)の、ロケーション P_x と Q_x は、ジョブ x を実現するためのプロセス P，Q の振る舞いを表す。具体的に、P_x は、初期状態が A_x あり、イベント req_x（当該イベントの発火はジョブ x の受理を表すようなイベント）が入力される(発火可能となる)とロケーション B_x に遷移し、イベント $a_x(20,*)$と

$a_x(0,*)$が XOR 関係で生成され(図では省略)、ロケーション B_x で b_x が入力される(発火可能となる)と クロック t_x がリセットされて A_x に遷移する。ロケーション Q_x も同様であり、P_x と Q_x の動作は互いに協調しながら同時並行的に進んでいく。また、ロケーション P_y と Q_y はジョブ y を受けたプロセス P,Q の振る舞いであり、ロケーション P_z と Q_z はジョブ z を受けたプロセス P,Q の振る舞いである。従来の時間ステートチャートでは、複数のジョブが並列処理される振る舞いを記述するためには、それら複数のジョブが例え同じ振る舞いをする場合であっても、各ジョブの実現に関わるプロセス同士が正しい組み合わせで(P_x と Q_x、P_y と Q_y、P_z と Q_z)インタラクションを取れるようにするための明示的な記述が必要であった。つまり、Fig.3.3-(1)のようにジョブごとに別々にロケーションやイベントを振る舞いを区別して記述する必要があった。

一方、同じ振る舞いを ETSC で記述した場合は、同じ振る舞いをする複数ジョブに対し、振る舞いの記述を一つ与えればよいため、Fig.3.3-(2)のように簡潔に書ける。このように簡潔に書ける理由は、時間ステートチャート上、ロケーション，イベント，タイマーに明示的に付されているジョブを識別するための添え字の機能を、ETSC では、ジョブ識別子を持たせたトークンと、その識別子を解読してイベント操作するイベントプール・マネージャによって実現しているためである。



(2) 拡張時間ステートチャートによる記述

(1) 時間ステートチャートによる記述

Fig.3.3: ジョブ識別子の効果

[例 3.3] (イベントプールとジョブ識別子で拡張された時間ステートチャートの動作)

Fig.3.4 は、イベントプールマネージャによるイベント操作、及び、IDJ のための基本操作に基づく拡張時間ステートチャートの振る舞いを説明するための図である。具体的には、Fig.3.2 の拡張時間ステートチャートの振る舞いの一部を示している。

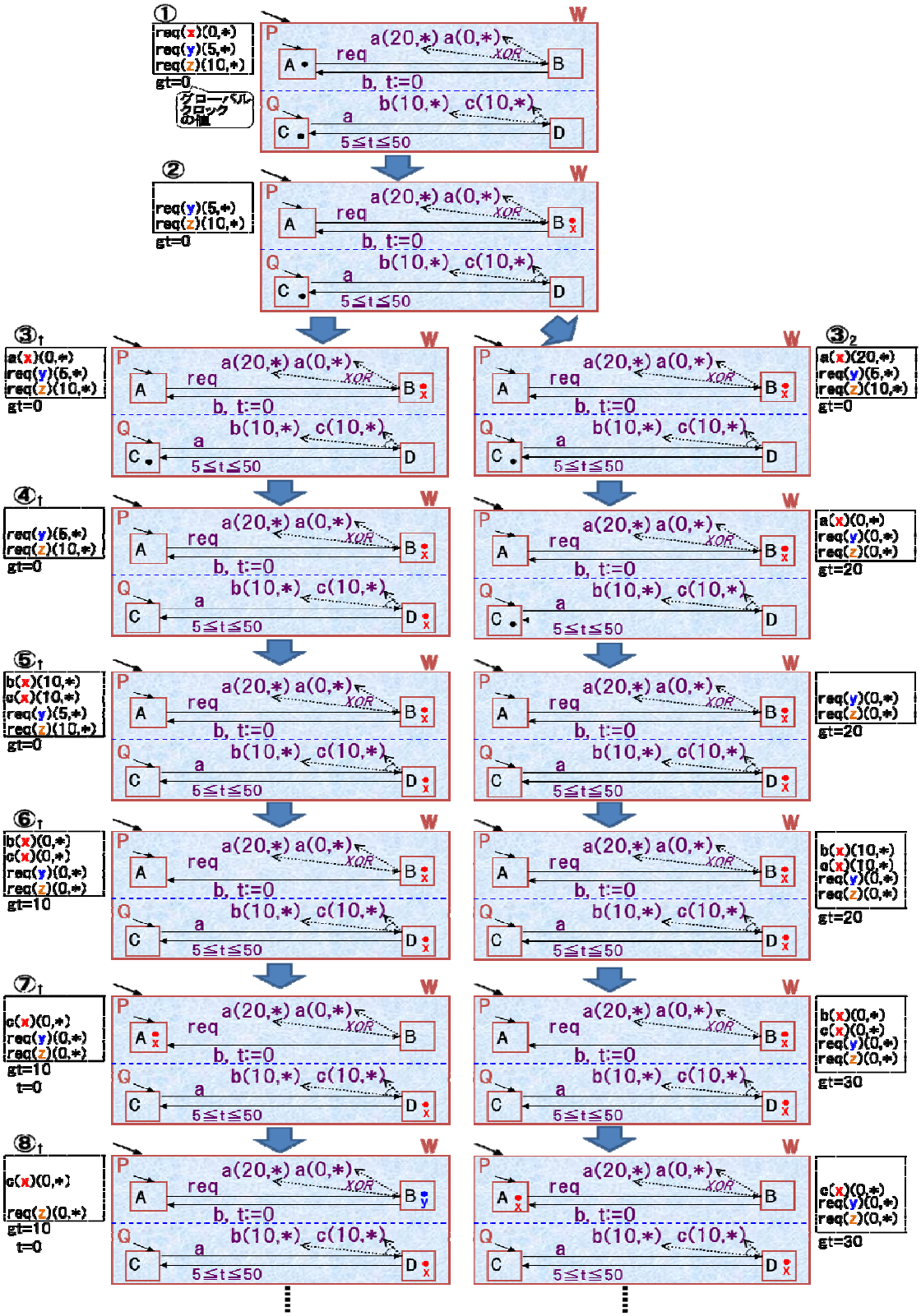


Fig.3.4: イベントプールとジョブ識別子で拡張された時間ステートチャートの動作

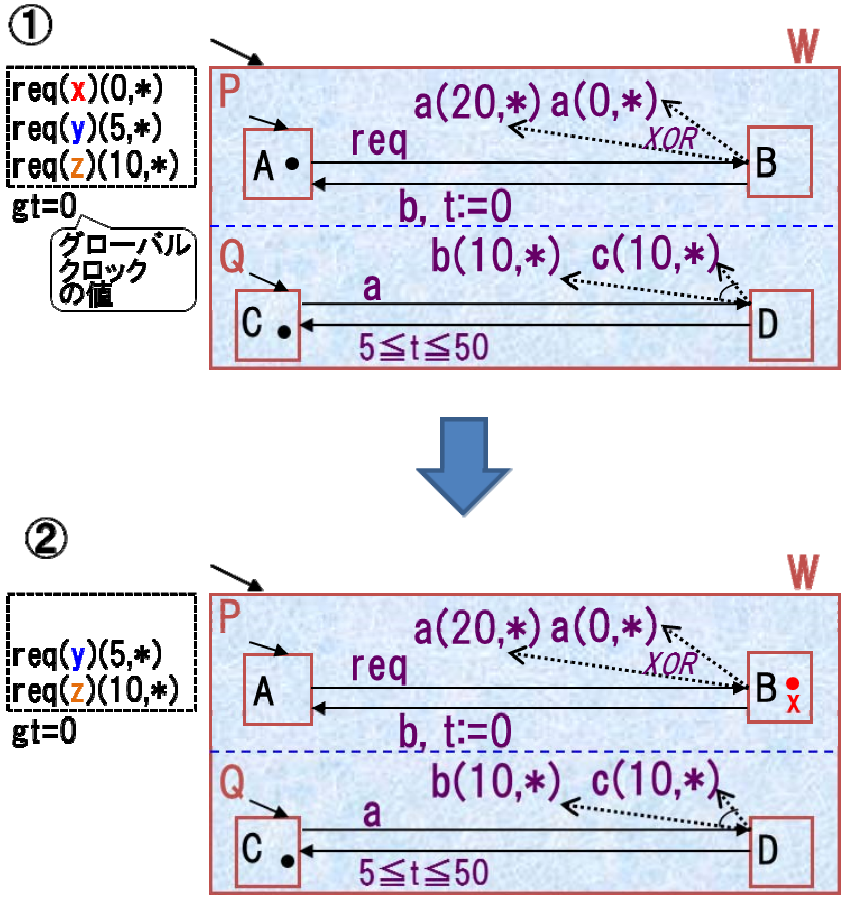


Fig.3.5:　IDJ “x” を持つイベント “req” の発火を伴う(A,C)から(B,C)への遷移

① 拡張時間ステートチャートの見方は例 3.1 で説明した通りである。破線四角はイベントプールを表している。内部に登録されている 3 つのイベントは、実行前に予め登録されている初期イベントである。一つ目のイベント req(x)(0,*)において、“req” はイベント名、()内の x はジョブ識別子(IDJ)、(0, *)は「0 クロック以降発火するまでずっと発火可能」といった発火可能期間を表す。全体の意味は、「イベント “req” で呼び出される製品で、製品タグ x を持つものの製造を、0 時間以降の可能なタイミングで開始せよ」となる。req(y)(5,*), req(z)(10,*)に関しても、IDJ と発火可能期間が異なる以外、同様の意味を持つ。

② イベント req(x)は直ちに発火可能であることから、イベントプール・マネージャによりシステム全体にブロードキャストされる。拡張時間ステートチャートの初期ロケーションは(A,B)であることから、イベント req は発火し、ロケーション A から B への遷移が起きる。このときイベント req に付された IDJ “x” は、遷移先のトークンに付与される。このように、遷移先のロケーションに、トークンのみならず、当該遷移を引き起こす原因となったイベントに付された IDJ も持

たせることの効果は、ロケーション B が現在アクティブ（処理中）ということだけではなく、現在どの製品を処理中かということまで認識可能になるということである。なお、発火したイベント a(x)(0,*)はイベントプールから抹消される。

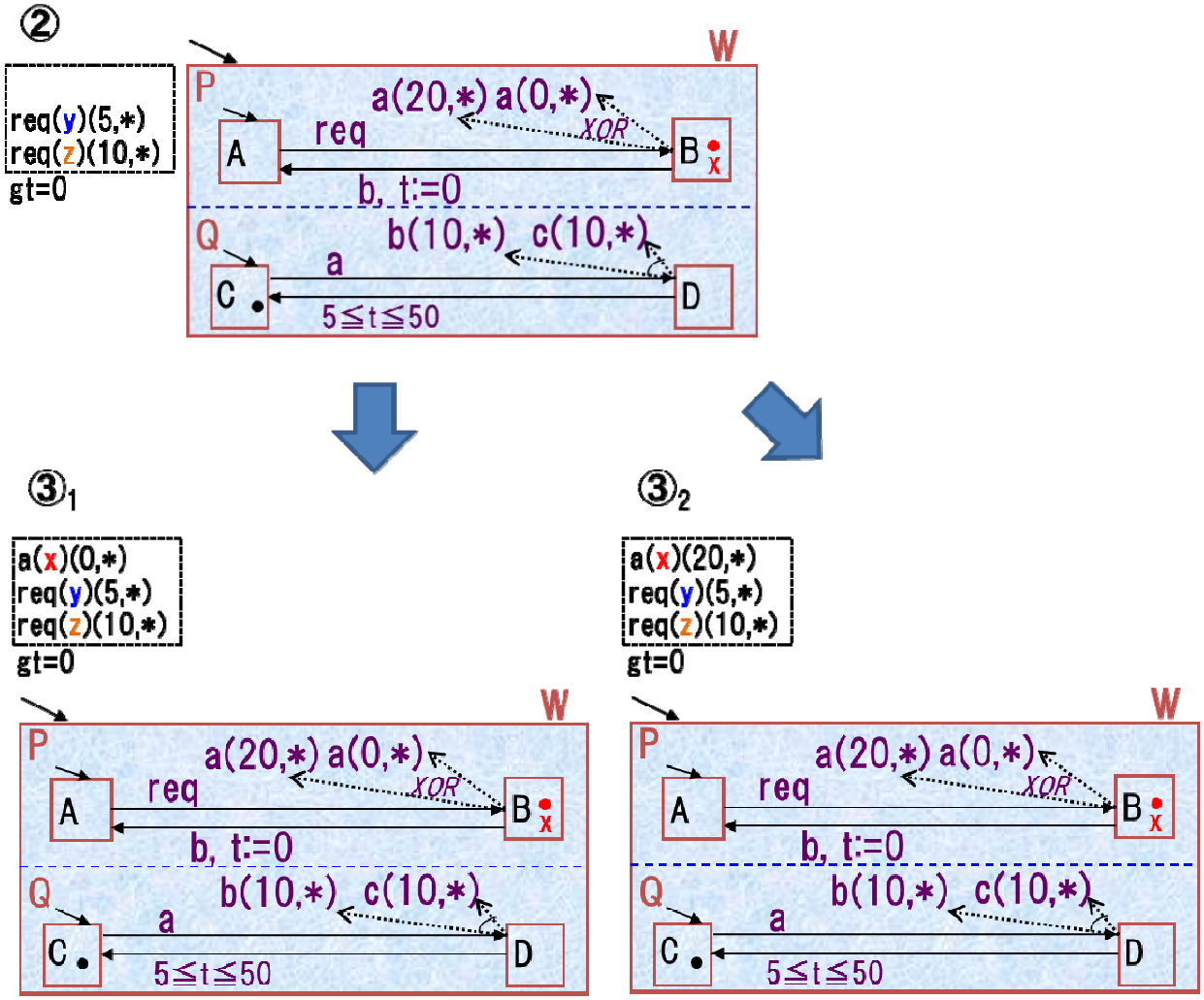


Fig.3.6:　XOR 型のイベント a(20,*),a(0,*) の生成に伴う分岐パスの生成

③$_1$ ロケーション A から B への遷移の直後、イベント生成則に従い、2 つのイベント a(0,*)と a(20,*)が、互いに排他的 OR の関係で生成される。また、2 つのイベントには、生成元のロケーション B が持つ IDJ “x”を付加し、a(x)(0,*)と a(x)(20,*)となる。なお、これらの 2 つのイベント a(0,*)(x)と a(20,*)(x)の間の XOR 関係を実現するため、次の状態は、次のようにして得られる。まず、②と同じ拡張時間ステートチャート(ETSC)を独立に 2 つ用意し、各々の ETSC に②のイベントプールを持たせる。(各々、③$_1$, ③$_2$とする。) そして、上記 2 つのイベントを別々に登録

する。具体的には、③$_1$の方のイベントプールへ a(0,*)を、③$_2$の方のイベントプールへ a(20,*)を登録する。こうすることにより、以降、2 つのイベント a(0,*)と a(20,*)が、同じ実行パス上で処理されることは起きない。つまり、これら 2 つのイベントの XOR の関係は実現される。

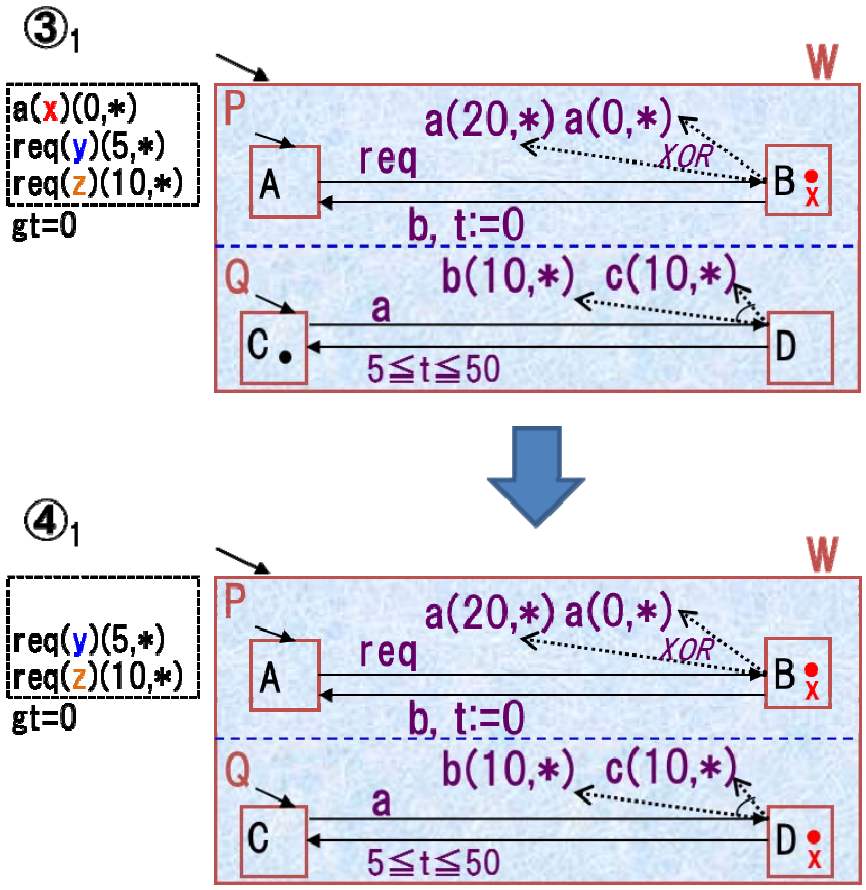


Fig.3.7: IDJ “x” を持つイベント “a” の発火を伴う(B,C)から(B,D)への遷移

④$_1$　③$_1$の次のステップは次の通りである。まず、イベント a(x)(0,*)が直ちに発火可能であり、そのことがシステム全体にブロードキャストされる。一方、ETSC 上アクティブなロケーションは(B,C)である。そこで、イベント a が発火し、ロケーション C から D への遷移が起きる。このときイベント a に付された IDJ“x”は、遷移先のトークンに付与され、発火したイベント a(x)(0,*)はイベントプールから抹消される。

（　③$_2$の次の状態④$_1$も同様の方法で得られる。）

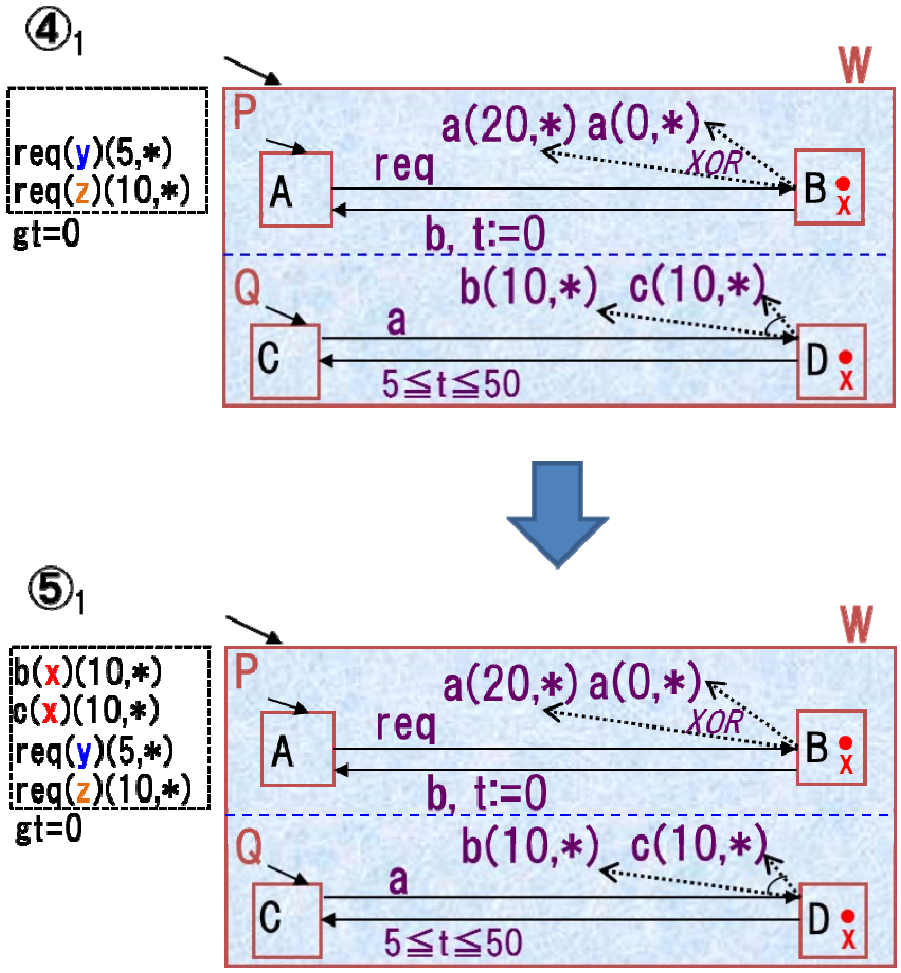


Fig.3.8: (B,C)から(B,D)への遷移直後のイベント b(10,*),c(10,*) の生成

⑤1 ロケーションCからDへの遷移の直後、イベント生成則に従い、2つのイベントb(0,*)とc(10,*)が、互いにANDの関係で生成される。また、2つのイベントには、生成元のロケーションDが持つIDJ “x”を付加し、b(x)(0,*)とc(x)(10,*)となる。なお、これらの2つのイベントa(0,*)(x)とa(20,*)(x)の間のAND関係を実現するために、これらは同じイベントプールへ登録する。

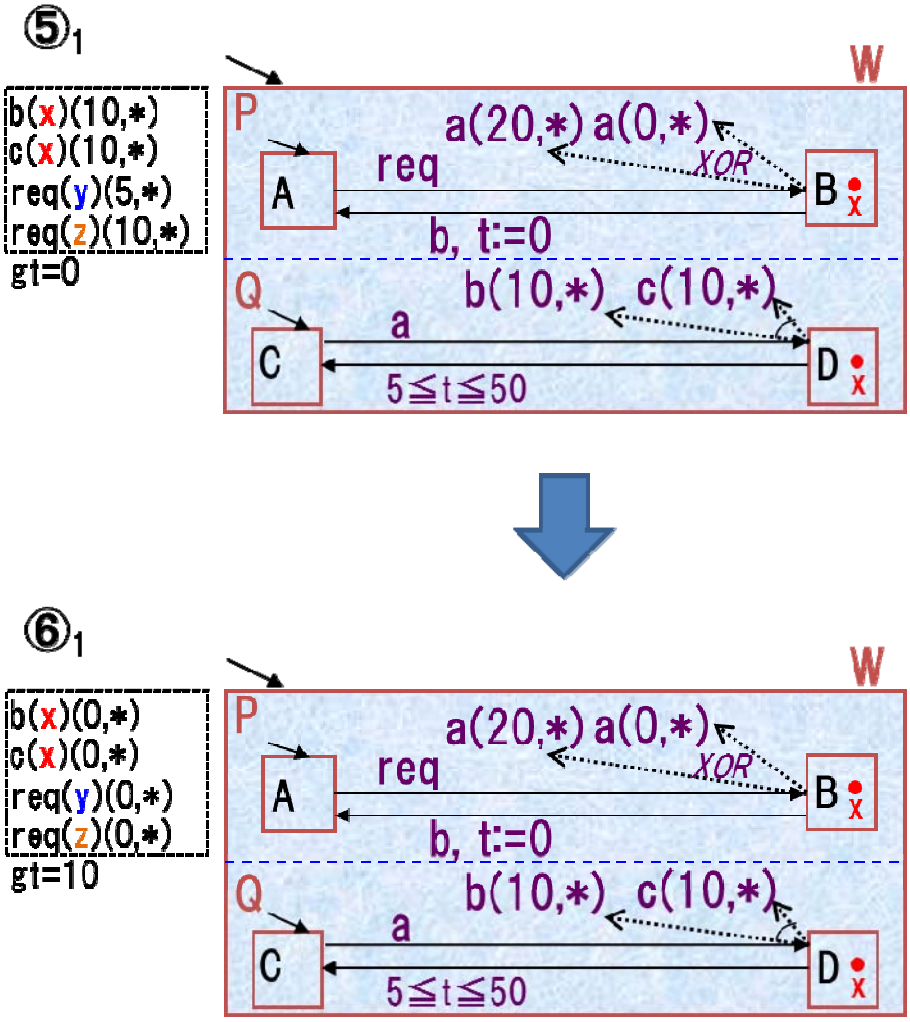


Fig.3.9: 何れかのイベントが発火可能になるまでグローバルクロックを進める

⑥$_1$　⑤$_1$ のイベントプールには直ちに発火可能なイベントはない。このような場合、次に発火可能なイベントが現れるまでグローバルクロックの値を進める。本例の場合、5 クロック後に req(y) が発火可能となる。しかし、ロケーション A がアクティブでないため、イベント“req”は発火できない。そこで、更に別のイベントが発火可能となるまで時間を 5 クロック進める。この時、グローバルクロック gt の値は 10　となり、イベントプール内の全てのイベントが「直ちに発火可能」となる。

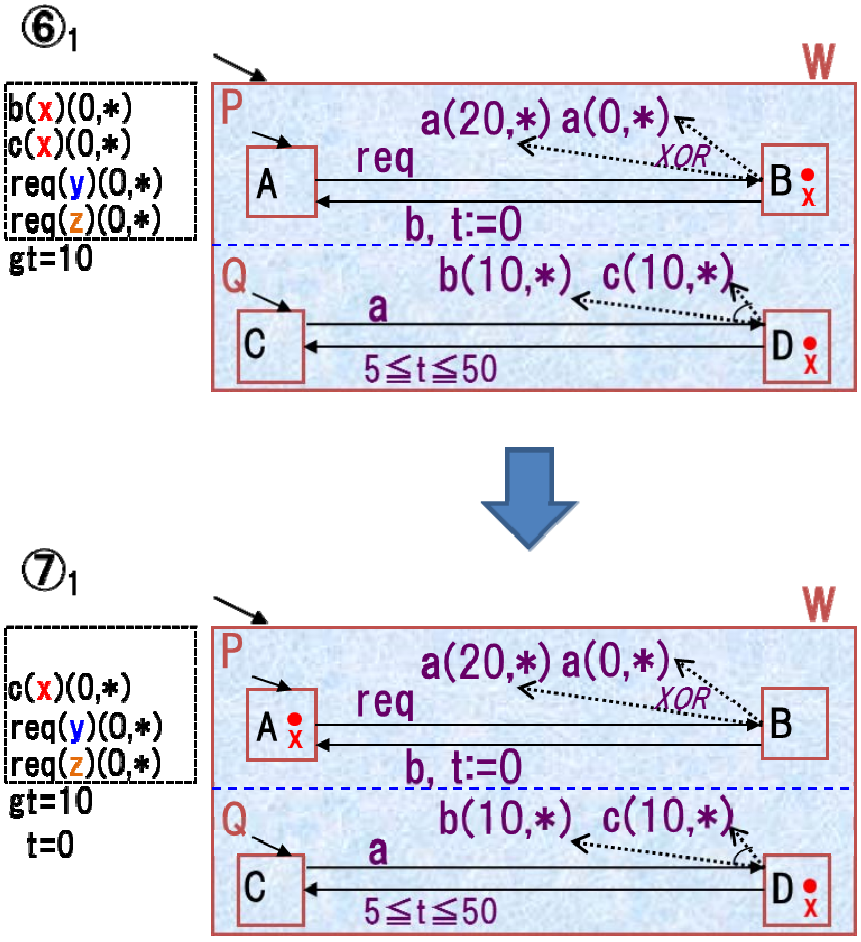


Fig.3.10: IDJ“x”を持つイベント“a”の発火を伴う(B,D)から(A,D)への遷移

⑦1　⑥1 のイベントプール内の全てが直ちに発火可能ということが、システム全体にブロードキャストされる。一方、現在アクティブなロケーションが B, D であることから、イベント b のみが実際に発火できる。そこで、b が発火し、ロケーション B から A への遷移が起きる。このとき、同時に、タイマー変数 t がリセットされ、t=0 となる。同時に、イベント b はイベントプールから抹消され、b に付されていた IDJ“x”は、遷移先 A のトークンに付与される。

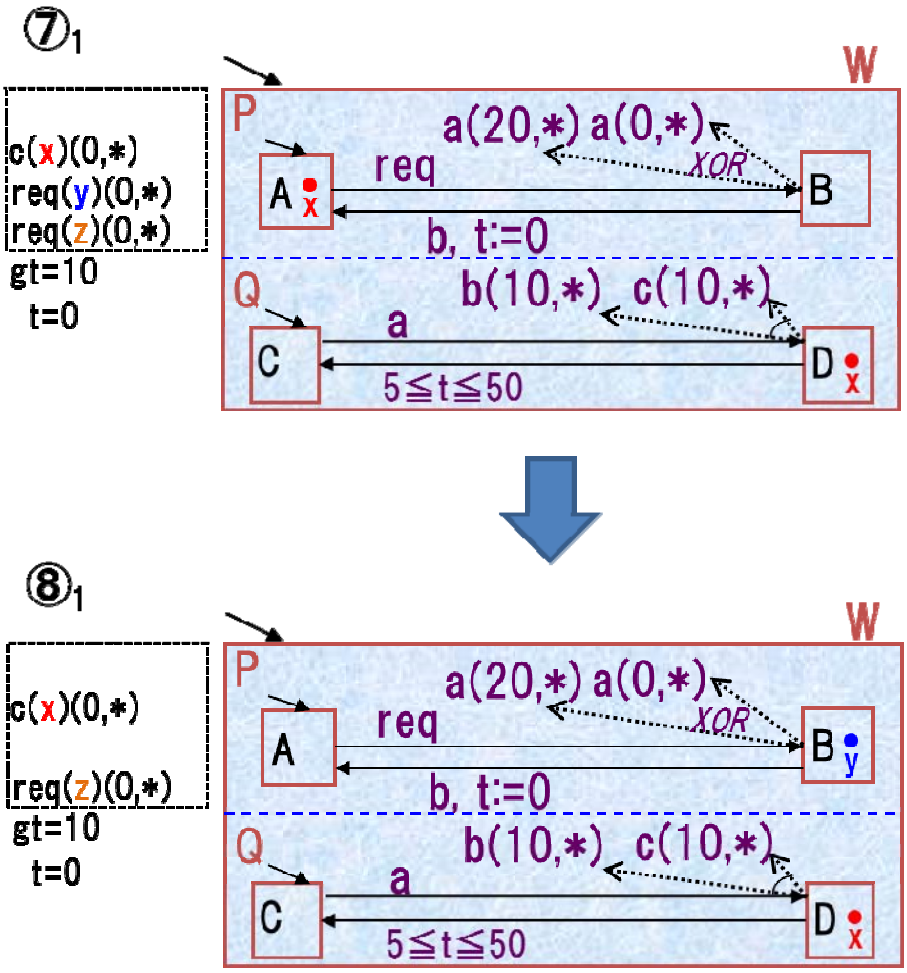


Fig.3.11: IDJ “y” を持つイベント “req” の発火を伴う(A,D)から(B,D)への遷移

⑧1　⑦1 のイベントプール内の全てが直ちに発火可能ということが、システム全体にブロードキャストされる。一方、現在ロケーションが A アクティブであるため、イベント req が発火できる。ここでイベント req として、req(y)と req(z)もの両方とも発火可能だが、一回の遷移に二つのイベントは発火できない。このような場合、どちらを発火させるかに関する取り決めは、実行前に予め決めておく必要がある。(Ex. ランダムに発火させる、別途与えられる別な制御ルールに従う、等)ここでは、req(y)(0,*)を発火させている。その結果、ロケーション A から B への遷移が生起し、イベント req(y)(0,*)はイベントプールから抹消され、req に付されていた IDJ “y” は、遷移先 B のトークンに付与される。

(以下、同様の処理を繰り返す。)

以上の例から分かるように、本手法では、導入した IDJ の効果として、同時刻において 2 種類のジ

ョブが並列処理されている様子や(⑧$_1$)、処理されるジョブの種類が入れ替わる様子が(⑦$_1$ から ⑧$_1$) 明確に表現できる。また、このような複数ジョブの並列動作や、内部イベントと外部イベントの混在が、イベントプール・マネージャに与えられた形式的な操作手順により、簡潔に扱われていることが分かる。

3.3 拡張時間ステートチャートの形式的定義

3.3.1 拡張時間ステートチャートの構文

[定義 3.1] (拡張時間ステートチャートの構文)

ETSC は以下の 12 項目で定義する。

・L：ロケーションの有限集合

特に最上位のロケーションを root∈L とする。

・A：アクション(イベント)の有限集合

(Af⊆A：発火イベント、Ag⊆A：生成イベント)

・IDJs：ジョブ識別子(IDJ)の有限集合

j∈IDJs が付随したロケーション l∈L を lj とし、IDJ 付きロケーションの集合を Ls とする。また、j∈IDJs が付随したイベント a∈A (或いは、a∈Af、a∈Ag)を aj とし、そのような IDJ 付きイベントの集合を AJ (同様に、AJf、AJg)と書く。

・C：タイミング制約式のためのクロックの有限集合

・C^2：生成イベントの発火可能タイミング制約式のためのクロックの有限集合

・B(C)：クロック C のタイミング制約式 δ であり、クロック変数 X と整数の時刻定数 d により再帰的に定義される。

$$\delta ::= X<d \mid d<X \mid \neg \delta \mid \delta 1 \wedge \delta 2$$

ここで、クロック変数 X∈C をリセットするタイミングは、特定の遷移関係の記述中のリセット式として明示的に与える。

・AE⊆Ag：イベントプールに蓄えられているイベントの集合 AE の要素 ae∈AE は、ae＝(a，ff，idj)で定義される。

-a は生成イベント(a∈Ag)

-ff は当該イベント a の発火可能タイミング制約であり、クロック変数 X と整数の時定数 d1，d2 により次のように定義される。

$$\delta ::= d1 \leqq X \leqq d2 \quad (0 \leqq d1 < d2 < \infty)$$

ここで、クロック変数 X∈C2 は、イベント生成時に当該イベントに持たせると同時にリセットする。

-idj は当該イベントの生成もとのロケーションが当該イベントが生成された時点で保持していた IDJ

・E：遷移関係の集合

$E \subseteq Ls \times 2^{Afj} \times B(C) \times 2^{C} \times AE \times Ls \times 2^{Agj} \times AE$　(lj1, evs1, fb, rs, aes1, lj2, evs2, aes2)∈E

- lj1∈L は遷移元ロケーション

- evs1⊆Af は発火イベント(アクション名)の集合

- fb∈B (C)は遷移を実行可能となるための各クロック変数の現在値に対する条件式 (ガード条件)

- rs∈2^{C} は遷移を実行後にリセットされるクロック変数の集合

- aes1⊆AE は遷移前のイベントプールのイベント集合

- lj2∈L は遷移先ロケーション

- evs2⊆Ag は生成イベントの集合

- aes2⊆AE は遷移後のイベントプールのイベント集合

なお、aes2 は、遷移前のイベントプールのイベント集合から発火イベントの全て(evs1)を取り除き、生成イベントの全て(evs2)を追加したもの。即ち、以下の通り。

aes2={aes1-evs1}∪evs2

・ρ ：階層関数　h : L → 2^{L}

l∈L のサブロケーションを返す。あるロケーション l のサブロケーションとは、ロケーションの階層関係を木で表現したときに、l の子となるロケーションのこと

・ϕ ：ロケーションの型関数　　L → {AND,OR}

・ϕA：生成イベントの型関数　　Ag → {AND,XOR}

・L0⊆L：初期ロケーションの有限集合

3.3.2　拡張時間ステートチャートの動作

拡張時間ステートチャート(ETSC)の動作は、複数ジョブに対する種々のイベントが混在するイベント集合による遷移からなる実行列である。各遷移は、当該遷移のアクション(イベントの発火)が実行可能であり、同遷移のガード条件を満たすときに生起する。ここで、発火可能なイベントは、EPM によって特定される。遷移後は、遷移を実行後にリセットされるクロック変数にゼロがセットされると共に、発火イベントに付された IDJ

は[IDJ のための基本操作]に従って遷移先に付され、発火イベントは EPM により EP から抹消される。なお、ETSC の遷移による動作には、イベント集合の型が AND 型か XOR 型かにより 2 種類ある。

以下、Fig.3.2 を用いて “コンフィギュレーション” の定義と例(例 3.4)を示した後に、ETSC の動作例(例 3.5)を示す。

[定義 3.2](コンフィギュレーション)

ロケーション l∈L のアクティブなサブロケーションを返す関数 ρ :L→2^L を用いて得られる、ある時点でアクティブな最大の状態集合をコンフィギュレーションと呼ぶ。

[例 3.4] (コンフィギュレーションの例)

Fig.3.2 において、A と C がアクティブであったとする。この時、ρ (W) ＝{P,Q}、ρ (P)＝{A}、ρ (Q)={C} となり、その時点でのコンフィギュレーションは{P,Q,A,C}と書ける。

以降、各表示の仕方は次の通りとする。

・遷移する各状態はコンフィギュレーション(Conf)、イベントプール(EP)、及び、初期状態からの経過時間 “t” の 3 つ組として表示する。

・イベント名の右下の添字は IDJ を表す。例えば、“req1” と “req2” は、共に Fig.3.2 中のイベント “req” の発火を引き起こすイベントである。そして、各々の発火によって引き起こされるジョブの実行列の各ロケーション、及び、生成イベントには、各々の添え字（“1” または “2”）を与える。具体的には、req1 によって引き起こされる実行列上のロケーション B は “B1”、生成イベント a は “a1” とする。

[例 3.5] (Fig.3.2 で示す拡張時間ステートチャートの動作)

初期状態：((A,C)∈Conf, EP={req1(10,*), req2(30,*)}, t=0)}

状態 2：((A,C)∈Conf, Ep={req1(0,*), req2(20,*)}, t=10)

[イベント req1 の発火, 同発火を伴う遷移 A→B, 同発火イベントに付された IDJ “1” を遷移先 B 上のトークンへ付与, 同発火イベントの IDJ“1”を生成イベント a(0,*)と a(20,*)へ付与, EP へ a1(0,*)と a1(20,*) を XOR 関係で登録, EP から req1 を抹消]

状態 3-1：((B1,C)∈Conf, Ep={ req2(20,*), a1(0,*)}, t=10) XOR

状態 3-2：((B1,C)∈Conf, Ep={ req2(20,*), a1(20,*)}, t=10)

(状態 3-1 の続き)

[イベント a1 の発火，同発火を伴う遷移 C→D，同発火イベントに付された IDJ“1”を遷移先 D 上のトークンへ付与, 同発火イベントの IDJ“1”を生成イベント b(10,*)と c(10,*)へ付与, EP へ b1(10,*)と c1(10,*)を AND 関係で登録，EP から a1 を抹消]

状態 3-1-1：((B1,D1)∈Conf，Ep={ req2(10,*), b1(0,*), c1(0,*)}，　t=20))　(以下略)

3.3.3　拡張時間ステートチャートの意味

ETSC の「意味定義」は、以上のような「動作」の定義を与えた結果得られる、振る舞い全体(操作的意味)で与えられるものとする。

3.3.4　イベント系列の受理可能判定について

挙動モデルとイベントプールを用いると、対象プラントの動作のシミュレーションが可能となるので、例えば外生的に与えられたイベント系列(動作系列)が受理可能かどうかの判定が可能である。具体的には、あるイベント系列の下でのシミュレーションパスが最後まで辿られるならば当該イベント系列は受理可能であり、シミュレーションパスが最後まで辿られず途中で中断されるならば、当該イベント系列は受理されない、といった判定ができる。

3.4　3 章のまとめ

本章では、離散型並列生産システムの「動作的特徴」を挙げ、これら動作的特徴を明確に扱えるような挙動表現モデルを開発した。開発の仕方としては、離散型並列生産システムのモデル化に必要な要件を元々最も多く備えている既存モデル「時間ステートチャート」(TSC)をベースモデルとして導入し、その TSC で扱えない「動作的特徴」(複数ジョブの並列実行、制御方法の例外時の切り替え等)を扱えるように TSC を拡張するという方法をとった。具体的には、イベントの生成、発火可能期間、及び、ジョブ-イベント間の関係性を持たせるための“ジョブの ID”が扱えるように TSC を拡張した。以上の拡張の結果、離散型並列生産システムの振る舞いを明確に表現可能な挙動表現モデル「拡張時間ステートチャート」が得られた。

第 4 章　離散型並列生産システムのための制御系の系統的な設計手法の提案

本章では、3 章で提案した挙動表現モデル(以下“ETSC”)をベースモデルとして使用し、処理時間の不確定性に対してロバスト性を持つ「事象駆動型生産システム制御系のモデルに基づく設計のためのフレームワーク」を提案する[28)~30)]。具体的には、まず、記述上の個人差を無くす目的で、ETSC を限定したサブセット「R-ETSC」を導入する。この時、R-ETSC の導入を機械的な手順で行えるようにするため、プロセスネットワーク型のモデルを中間モデルとして利用する。また、時間駆動の情報として得られる目標スケジュールを、処理時間が多少変動しても利用可能な事象駆動の情報へ、系統的に受け渡す方法を提案する。具体的には、まず、制御に関する情報を、スケジュールによって動的に変わる部分と、スケジュールによって変わらない静的な部分とにモデルベースで切り分けて記述できるような仕組み(テンプレート)を導入する。更に、これらテンプレートに埋め込む情報を、R-ETSC からシステマティックな手順で抽出する方法を示す。更に、以上のようなテンプレートを利用する方法を使って、より大きな外乱により再スケジューリングが不可避となった場合においても、現在稼働中の対象プラントを止めることなく、その再スケジューリング結果を取り込み、スムーズにシステム運用を継続できるような方法を示す。

以下では、まず、本手法を考える上で最も大きな問題である「計画系と実行系のギャップ」に対し、本稿で採る方策について示した後、提案するフレームワーク全体の基本的考え方を示す。次に、本手法のベースモデルとして使用する ETSC の記述上の個人差を無くすために導入する限定したサブセット(R-ETSC)について示し、R-ETSC に基づく制御系実行モジュール生成のためのテンプレートとその利用方法について示す。更に、計画変更にも対応可能なシステム運用を可能とする制御系動作メカニズムについて示す。

4.1　計画系と実行系のギャップをなくすための方策について

4.1.1　計画系から実行系への動的な情報伝達の問題について

計画系での問題解決(スケジューリング)に関しては、MILP や DP のような OR 的手法やその他の数理計画法を用いる手法や、そのような数理的扱いが困難な複雑な構造を持つ場合にはヒューリスティクス的手法が、様々に研究・提案されている[44)~47)]。しかし、いずれの場合も、その解は時間駆動の操作の列として与えられる。一方、実行系での制御の実装では、ペトリネットを使った方法[48)]、マトリックスベースの手法[49),50) 49),50)]、FB(Function Block)[14)~17)]を使った方法、或いは、バッチ型化学プラントでの

国際標準規格 S88[22]を使った方法等、様々な実装方法や開発支援環境が提案されている。しかし、いずれの方法においても、計画系から実行系への動的かつ円滑な情報伝達の問題に関しては、系統的な解決法が与えられているとは言えない。他の分野、例えば、業務・計画系と制御系を円滑に「つなぐ」べき情報システム MES（Manufacturing Execution System）等の分野では、近年、システムの標準化や部品化、あるいは各サブシステムの統合等に関する観点からの議論や試みがさかんに行われているが、本研究で扱っている生産ラインレベルでは、実際には、各企業が独自に蓄積してきたノウハウに基づき、適宜、差立や、制御プログラムの作成・追加・変更など、かなりの手作業を介してプラント運転しているのが現状である[18)～21)]。

4.1.2　計画系のスケジューリング結果を実行系へシステマティックに受け渡す方法での基本方針

スケジューリング結果を実行系へシステマティックに受け渡す方法の実現では、次の 2 つの考え方を基本方針とする。まず、一つは、制御に関する情報は、スケジュールによって変わらない静的な部分と、スケジュールによって動的に変わる部分とに分け、別々に記述するということである。このことは、5 種類のテンプレートを導入し使用すると簡単に行える。更に、もう一つは、これらテンプレートに埋め込むべき情報は、ETSC からシステマティックな手順で抽出するということである。

ここで、「スケジュールによって変わらない静的な情報」とは、プラントの装置特性に因って決まる操作手順、製造レシピ、プラントの安全性を維持するための制約、製品の品質を保証するための制約、などである。一方、「スケジュールによって動的に変わる情報」とは、スケジューリング結果からしか得ることが出来ない(事前に知ることが出来ない)情報を指している。このような情報として、本稿では、各処理の装置割り当てに基づく装置の使用順序を、スケジューリング結果が示す各イベントの時刻情報から抽出し、使用する。

[例 4.1](スケジューリング結果が示す「時刻情報」を「順序関係」に変換する方法) (Fig.4.1 参照)

Fig.4.1 は、装置 A～F を持つ生産プラントのスケジューリング結果を表すガントチャートの例である。セグメント内の番号(1,2,…は製品番号)を表す。本プラントでは、装置 A で処理された処理対象は、装置 B を用いて、装置 C,D,E,F の何れかに移送され、装置 C,D,E,F で処理された処理対象は、また、装置 B を用いて、装置 G へ移送される。つまり、装置 A,C,D,E,F から装置 B へ出される使用要求は競合する可能性がある。

以上のような「装置 B」を使用する順序は、Fig.4.1 に示すガントチャートから、次のようにして機械的な手順で抽出できる。

・装置 B の処理の開始時点(セグメントの左端)を終了時刻に持つような装置名(A/C/D/E/F)を順に抽出する。

実際、Fig. 4. 1 のガントチャートからは、装置 B に関する次のような使用順序が取り出せる。

［①装置 A，②装置 A，③装置 F，④装置 A，⑤装置 D，⑥装置 A，⑦装置 E，⑧装置 A，⑨装置 D，⑩装置 C・・・］

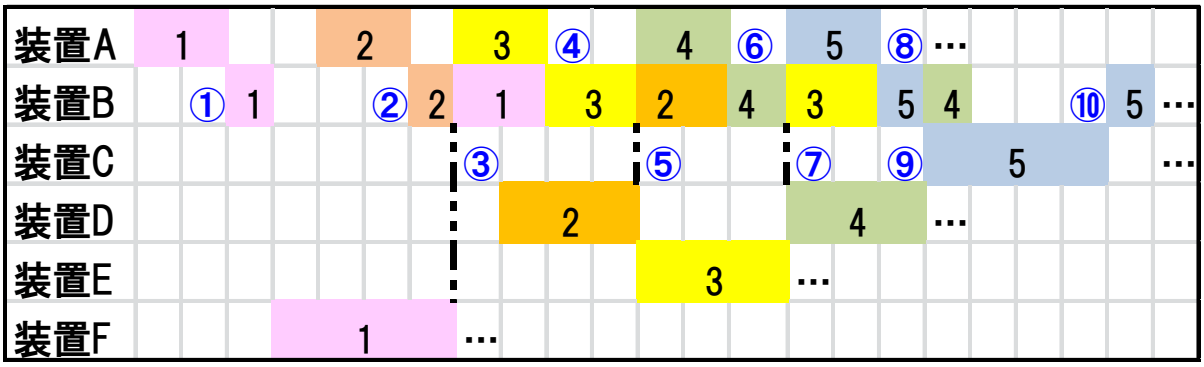


Fig.4.1:　スケジューリング結果が示す時刻情報を「順序関係」に変換する方法

以上のように、スケジューリング結果を順序情報の形で取り出す理由は、順序関係は、計画からの時間的ずれが発生しても直ちに利用できなくなるということはないため、そうすることによって「小さな不確定性変動の影響を、スケジューリング結果の持つ最適性をあまりそこなうことなく吸収することができる」からである。ただし、遅れの累積分、計画全体が後ろにずれていくことにより効率低下や制約違反が生じる可能性があるため、ある時点で再スケジューリングが必要となることがある(注 4. 1)。本稿で提案する制御システムでは、そのような再スケジューリング結果に基づく制御の変更を対象プラントを止めずに実行できる、ということを重要な要件としている。なお、対象プラントの制御動作の実現では、制御のためのパラメータを読み込んだ制御プログラムをコントローラに逐次解読させ、その時点で実行すべきプログラム群をマルチスレッド方式で並列処理することにより、実機の各部分の並列動作を実現するという方法を取る。

(注 4. 1) 本稿では、再スケジューリングすべきタイミングは「外生的に与えられるもの」とし、そのタイミングの特定の仕方に関しては、本稿の中では言及しない。

「再スケジューリング」に関する研究分野として、初期スケジュールに基づく実行中、適当なタイミングでスケジュールを修正するといったアプローチを取る「リアクティブ・スケジューリング」と呼ば

れる分野があり、多くの研究成果が報告されている。しかし、スケジュール修正のタイミングの意思決定は、実際には、定期的な再スケジューリング以外では、専門家の経験的知識に依存、個々のシステムや場面に特化した対応、といった方法を採っているのが現状である。これまでの研究においても、不確定な事象が発生し次第、再スケジューリングを行うことを前提としていることが多い。そのため、実際には、スケジュールの実行可能性をほとんど妨げることのない(即ち、システムが吸収できる程度の)不確定変動であってもスケジュールを修正するような事態を招くことがある。

一方、「再スケジューリングを行うタイミングを特定するための方法論」に対する検討事例は少ないが、マッチアップの考え方に基づき限界遅延タスク数を特定するというシミュレーションベースでの判断方法が提案されている。また、計算機代数の理論である QE（Quantifier Elimination）を使い、離散型生産システムが要求仕様を満たしながら実行できる実行可能領域を計算する方法も提案されている。

4.2　ETSC に基づく制御系設計手法の基本的考え方

本稿で提案するフレームワーク(Fig.4.2)は A,B2 つのステージから成る。以下、各ステージについて説明する。

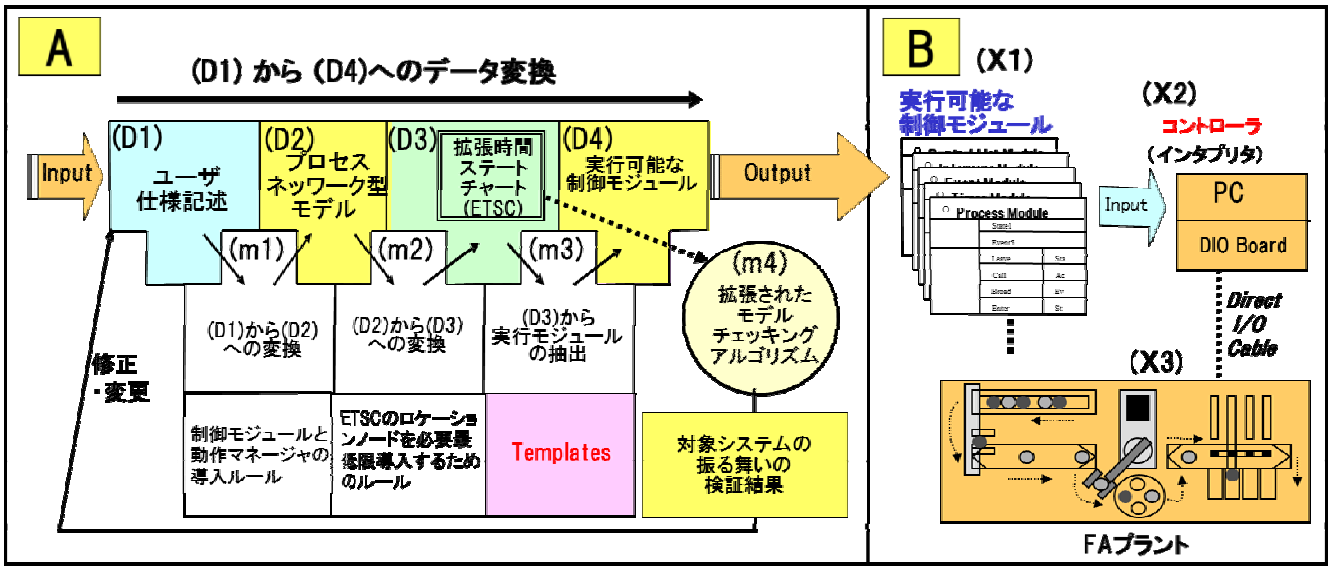


Fig.4.2:　ETSC に基づく制御系設計手法のフレームワーク

4.2.1　制御リストの準備

まず、ステージ A の前段階として、最適スケジューリングの結果得られる時間駆動の要求動作を、実行系で使用できる事象駆動の動作(順序情報)に変換する。ただし、スケジューリング結果(ガントチャー

ト)には、装置構造に起因する制約や製造レシピ等、多種多様な情報が埋め込まれているので、どれを制御のために使用するかに関する選択の基準が必要である。

本方法では、生産計画は頻繁に変わるということを考慮し、「スケジューリング結果からしか取り出せない情報」のみを取り出すという選択基準を与える。具体的には、各ジョブに対する「装置割り当て」と「装置使用順序」のみ取り出すとする。また、同じく変更時の効率を考慮し、スケジューリング結果から取り出す情報に関しては、その記述も扱いも簡単にする。具体的には、各装置に関し、当該装置を使用するジョブ全体での使用順序を表すリスト(以降“制御リスト”)として記述するとする。そして、その使用順序の変更も、制御リストの要素の追加や更新のみで行えるようにする。

4.2.2　ステージ A について

以上のように、本方法では、スケジューリング結果から取り出す情報に関しては、その変更が頻繁であるが故に、出来るだけ扱いを簡単にできるようにしている。しかし、その場合、スケジューリング結果に左右されない制御部分、即ち、装置構造や製造レシピに起因する各種制約を守らせるための制御手順の記述が複雑になりがちである。この問題に対し、本稿で提案するフレームワークでは、プラントの構造的特徴を直に反映できる“プロセスネットワーク型のモデル”を中間モデルとして用いることにより、問題を回避する。

Fig.4.3 は、プロセスネットワーク型モデルの概念図である。プロセスネットワーク型モデルとは、プロセスをノードとし、処理（信号）の流れをアークとするネットワーク構造型のモデルである。データの流れが実行の流れを制御する「データフロー同期機構」を持つ。また、プロセスネットワーク型モデルの特徴は、何らかの制御を要する箇所がネットワークの構造的特徴として現れるようなモデル化が可能である。具体的には、複数のアークの根元が競合する構造部分を Dispatch 型の構造と呼び、その部分には、1 本の処理系列を複数プロセスへ分配するような制御モジュール（“Dispatcher”）を与える。また、複数のアークの先端が競合する構造部分を Merge 型の構造と呼び、その部分には、複数プロセスからの処理を 1 本の処理系列にマージするような制御モジュールを与える。

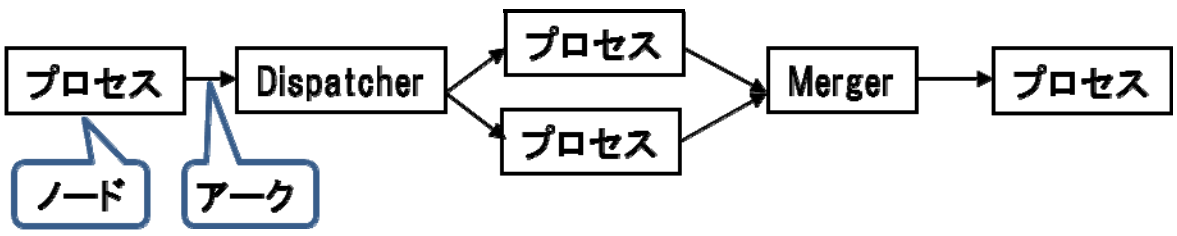


Fig.4.3:　プロセスネットワーク型モデルの概念図

以上のようなプロセスネットワーク型モデルを用いることにより、この種の制御系設計で特に問題となる競合解消や排他制御といった何らかの制御を要する箇所の特定を、ネットワーク構造の図的特徴(アークの先端、或いは根元が結集する等)が現れる部位として明確に特定することができる。

以上を踏まえ、ステージ A では、まず、初期データであるユーザ仕様記述(プラント仕様、制御仕様)をもとに、中間モデルであるプロセスネットワーク型モデルを作成し、挙動モデルである ETSC を作成する。次に、制御に必要な情報を ETSC から要約・抽出するため、5 種類の分類に基づく「テンプレート」を用意し、そのテンプレートに基づいて実行モジュール群を抽出生成する。こうして得られるランタイムモジュールである実行モジュール群が、制御実行部 B ステージへの入力データとなる。
ここで、挙動表現モデルとして ETSC(Fig.4.2-(D3))を用いる利点は、本フレームワーク実現の核となるプロセスネットワーク型モデル(Fig.4.2-(D2))とスケジューリング結果である順序情報を統合でき、更に、そこから、最終的に生成すべき制御モジュール群(Fig.4.2-(D4))にとって必要な情報を要約・抽出させるという一連の操作を効率良く行えるということである。以上のような ETSC の利点は、双方のモデル(プロセスネットワーク型モデルと制御モジュール群)との対応関係が付け易い ETSC の表現形式に因る。

なお、ユーザ仕様記述をプロセスネットワーク型モデルへ変換するための手法(図中(m1))に関しては、そのガイドラインが開発されており [54]、また、プロセスネットワーク型モデルから ETSC への変換(Fig.4.2 (m2)) のための形式的手順も、既に開発し、変換プログラムとして実装している [28] 。本研究では、Fig.4.2 のフレームワーク全体を完成させるために、ETSC から制御系実行モジュール群への変換(Fig.4.2 (m3))のための形式的手順の整備と、それを実現するための変換プログラムの開発・実装を行った [29]。

なお、ETSC モデルを用いて制御モジュール生成を行うことによって得られる利点は次の 2 つである。まず、一つは、このモデルを使うと制御情報の抽出が機械的に行えるということである。ここで「機械的」というのは、モデルの構造や記述形式のみから判断し、意味を考えることなく情報抽出できるということである。そのようなことが可能となる理由は、ETSC 挙動表現モデルの構造と記述形式にある。具体的には、まず、制御情報の基本単位は、一対のロケーション間の遷移構造として機械的に抽出できる。更に、階層上それ以下の各種制御情報は、各遷移構造に付随する遷移条件及び出力イベントとして与えられるため、系統的な情報抽出が可能である。ただし、このような遷移条件や出力イベントが与える情報には、時間制約、タイマー制約、割り込み処理、装置割り当て等のスケジュール情報といった、都合 4 種類の制約が混在している。しかし、いずれの制約の記述も、各性質特有の文字列や符号を含ん

でいる。そのため、いずれの制約記述も、4 種類の何れであるかの識別を、形式的な手順で行えるようになっている（4.5.2 で後述)。なお、以上のような情報抽出を、振る舞いの種類ごとに用意するテンプレートを使って、種類ごとに“漏れなく行える”仕組みは、本稿で提案する「ETSC 上の記述と実機動作との対応関係を付け易くするための記述上の制限」(4.3.2 節で後述）によって与えており、そのような情報抽出操作の具体例は、4.5.2 節で示す。次に、ETSC モデルを用いることによって得られる 2 つ目の利点は、スケジュールとそれ以外の情報の切り分けが可能になることである。ここで、その理由は、上述したように、ETSC で作成した対象システムのモデルでは、混在する 4 種類の制約情報を、特定の文字列や符号を手がかりに、機械的に切り分けが可能となるからである。なお、本稿では、以上のような特徴を持つ ETSC を使用することにより、スケジュール情報のみを抽出し、スケジュール内容を更新し、それを元のモデルの所定の位置に戻すことを、“テンプレート”を使って機械的に行えるようにした。その結果、再スケジュールに切り替える際に必要となる制御システム変更の手間を最小限に抑えることが出来た。具体的に言い換えると、スケジュールが変わっても、それに必要な部分だけ取り替えれば、それ以外の制御構造は変えずに（即ち、シーケンスプログラム事態は書き換えなくても)、パラメータのみ書き換えればよいということである　(4.1.2,4.4.1 節)。

4.2.3　ステージ B について

ステージ B は、A で得られた実行モジュールを用いて対象プラントの制御動作を実現するステージである。ここで、制御対象内の各装置やセンサには、それを駆動するためのドライバープログラムが備わっている。制御の実行は次のようにして行われる。まず、上記のドライバープログラムに、ステージ A で作成されたコントローラへの入力データである実行モジュール群((Fig.4.2-(D4))を埋め込む。そして、実行時にはそのドライバープログラムをコントローラがインタプリタ方式で逐次解読し、次に実行すべきプログラムの集合とパラメータの値を特定する。更に、それらパラメータの値を伴うプログラムの集合をマルチスレッド方式で並列処理することにより、実機の各部分の並列動作を実現する、という方式である。

4.2.4　再スケジューリング時の対処方法

以上、外乱（遅延などの不確定変動）が小さく、現在組み込まれている制御のまま実行が継続可能なときの制御の実現方式は 4.2.1～4.2.3 節で述べたとおりである。一方、現行の制御のままでは吸収しきれないような大きな外乱発生等のため再スケジューリングが行われた場合は、4.2.1 節で述べたように、

制御リストの変更のみで対処できる。しかし、そのような制御リストの変更をプラントを止めずに新旧リストの単純な差し替えで行うことは、両リストの間での同期を取ることが難しく困難である。そこで、実装上の工夫として、リスト間の同期を考慮せずに制御リストの変更を行える、次のような方法を採った。まず、再スケジューリング結果に対して作成された制御リストを、既に与えられている 制御リスト の末尾に単純に付け加える。そして、この時生じる新旧 2 つのリスト間の要素の重複部分(即ち、再スケジューリング対象となった領域に対応する部分)に対して、古い方のデータについては強制的に「既実行(Flag = 0)」とし、新しいデータについては「未実行(Flag = 1)」としておくことにより、その重複を回避する。制御動作をリストとフラグで管理するこのような方法を取ることにより、実行中のプラントを止めることなく、新たなスケジューリング結果を反映した制御へ動的かつスムースに切り替えることが可能となる。

4.3 制御系設計のための挙動表現モデル

冒頭で述べたように、本設計手法では、本論文の 2 章で提案する挙動表現モデル ETSC [43] をベースモデルとして使用する。

ETSC について簡単にまとめると次のようになる。ETSC は、既存モデルの持つ階層性と並行性の表現能力や、事象駆動と時間駆動が混在した複雑な振る舞いの表現能力は引き継ぎつつ、内部生成イベントの記述文法ならびにモデルの実行セマンティクスを与える機構を新たに導入することによって、発火可能期間を持つような個々の内部生成イベントの制御を容易に行えるように拡張したものである。また、ジョブ間、或いは、ジョブ-イベント間の関係性を与える “ジョブの ID”を持てるように拡張されているため、各ジョブに対する並列動作のモデル（同じ振る舞いごとにまとめて記述されるオートマトン）をジョブごとに個別に与える必要がない。そのため、モデルに持たせるべきロケーションやアークの数の増大化を防ぐことができ、ある程度大規模で煩雑な対象システムでも、その振る舞いを簡潔に表すことができる。

ただし、ETSC は、その表現力の柔軟さのために、モデル作成上の個人差が生じやすく、何らかの制限を設けない限り、それに基づいて開発された制御系にも当然個人差が生じることとなる。

この問題に対処するため、本章では、ETSC モデル作成にあたって、2 つの側面からの記述上の制限を導入する。具体的には、①プロセスネットワーク型モデルを ETSC の中間モデルとするという制限 [55] と、②ETSC 上の記述と実機動作との対応関係を付け易くするための記述上の制限である。プロセスネットワークを中間モデルとして対象の振る舞いの骨格を表すことにより、最終的に得られる ETSC モデ

ルの階層が 3 階層に統一される。また、②の制限を用いることにより、各装置の操作手順や工程の実行手順といった各種シーケンス情報を、ETSC のどの階層にどういう形で持たせるべきかが指定される。これらのことにより記述上のばらつきを抑えることができるため、個人差のない挙動モデルを得ることが可能となる。

以下では、まず、説明上使用する簡単な例「ミニ FA プラント」(Fig.4.4)について示す。次に、その例を使って、上記①，②の制限について順に示す。具体的には、まず、プラント情報のユーザ仕様記述方法について示し、ユーザ仕様をプロセスネットワーク型モデルへ変換する方法について示す（制限①)。次に、ETSC に上記②の制限導入した上で、プロセスネットワーク型モデルから ETSC への変換方法について示す。

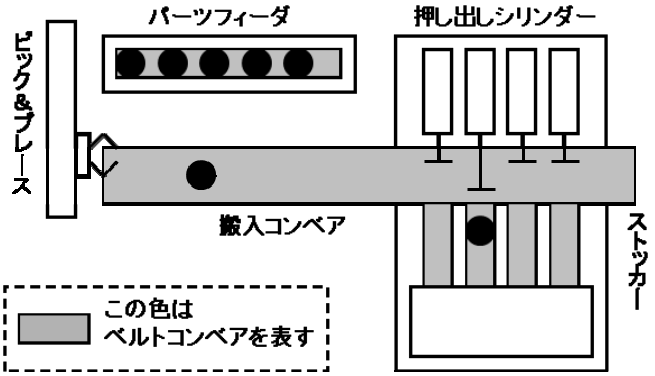


Fig.4.4:　ミニ FA プラント

説明のための例「ミニ FA プラント」

離散型生産システムの動作は、次の 4 種類の振る舞い特性によって定まる。

・プラント上に設置されている装置群の特性（装置モデル）

・プラントで製造する製品の製造レシピ（プロセスモデル）

・生産環境の不確定性に関する情報（不確定環境モデル）

・操作上の自由度の有る箇所に与える操作可能な諸パラメータ値（運用戦略モデル）

制御系設計のスタートである、以上 4 種類の振る舞い特性の記述は、事前にユーザによって行われる必要がある。本研究では、そのようなユーザ仕様記述のためフォーマットを定義した。また、定義するフォーマットを使用して得られるユーザ仕様記述からは、制御系設計の次の段階であるプロセスネットワーク型モデルの生成が、形式的な手順で行えるようになっている。

ミニ FA プラントの動作仕様は次の通りである。

・パーツフィーダには予め処理対象(ワーク)が左から処理順に並べられている。

・ピック＆プレースはパーツフィーダの左端にあるワークを、そのアームでつかみ上げ、搬入コンベア上へ移動させる。

・搬入コンベアはワークを右方向へ流す。

・押し出しシリンダーでワークをストッカーへ押し出す。

・ストッカー上で一定時間の加工処理が行われ、処理後、ワークは系外に押し出される。

4.3.1　ユーザ仕様記述をプロセスネットワーク型モデルへ変換するための方法について

１．プラント情報のユーザ仕様記述

以下では、幾つかの概念・用語を定義した後、4 種類のユーザ仕様記述（装置モデル、プロセスモデル、不確定環境モデル、運用戦略モデル）について説明する[36)]。

概念・用語の定義

・機能：　「機能」とはなんらかの「制御目的」を達成させることによって実現されるものと定義する。そして、このような機能が制御を行なう際の「着目範囲」とは、装置と一対一対応とは限らない。複数の装置を１つの機能の着目範囲とする場合や、１つの装置を複数の機能で重複して着目範囲とする場合もある。

・動作シーケンス：　「動作シーケンス」とは、4 つのモデル中にばらばらに埋め込まれている以下のような順序制約によって決まる動作の系列を意味する。

　・製造手順として与えられる順序関係

　・プラント制約として決まってくる順序関係

　・イベントの発生とその発火をトリガとして開始／終了するような動作との間に存在する順序関係

・機能単位：　以下の要件を満たすような動作シーケンスの集合を「機能単位」と呼ぶ。

　・各動作シーケンスは実行可能かつ、実行を開始させるかどうかの制御が可能

　・動作シーケンス同士は同時実行が不可

・挙動単位：　各機能単位に、以下１～３の操作を行う事で最終的に得られる「動作シーケンスの集まり」各々を、“挙動単位”と呼ぶ。各挙動単位は、プロセスネットワークの生成段階では各ノードにマッピングされる概念である。

（1）動作シーケンス集合の追加：

これは、いずれの機能単位にも含まれない動作シーケンスの各々を、動作マネージャ間の通信効率に

とって最良と思われる機能単位（動作シーケンス）に各々追加していくという操作である。

例えば、「ある処理の実行終了タイミングを示す動作シーケンスは、当該処理の実行開始タイミングを部分シーケンスとして包含するような動作シーケンスに追加する」という操作である。

（2）動作シーケンス集合のコピー

これは、一つの装置で同時に複数の処理対象が独立に処理可能である場合は、そのような処理対象の個数分、当該動作シーケンス集合のコピーを(互いに独立な関係として)作成するという操作である。この操作の目的は、プラントが元々持っている並列性をそのままモデルに反映させるためである。具体的には、プロセスネットワーク上、並列動作可能な処理のまとまり同士は互いに独立なノードとしてモデル化されるようにするためである。

（3）動作シーケンス集合間の統廃合

これは、対象プラントの動作の本質的な並列性を損なうことなく統合可能な 2 つの動作シーケンスがあれば、これら 2 つを一つの挙動単位として丸め込んでしまう操作である。この操作の目的は、ノードにマッピングされる挙動単位の数を最低限に抑えることにより、簡潔なプロセスネットワークを得るためである。

装置モデル

装置モデルではプラントを構成する各装置の特性や装置間の関係（“機能(※1)”を実現させる上での装置同士の依存関係）など、装置の仕様上確定している情報（ユーザの意図により変更できない情報）を定義する。

<< 装置モデルの定義項目 >>

E0.定義する“機能”の名前

（以下、この“機能”に関する定義項目）

E1.着目する装置の名前（複数定義可）

E2.当該装置において実現させようと考えている動作シーケンスの集合

(“機能単位”の集まり)

E3.当該装置において同時並行的に実現可能な（E2 で名前を定義した）動作シーケンス数

（E2 で名前を定義した各動作シーケンスに対してそれぞれ E4～E7 を定義する）

E4.他の機能によって決定される当該動作シーケンスの開始／終了条件

E5: 当該動作シーケンスの構成内容（事象の半順序構造）

（E5 で定義した各事象に対してそれぞれ E6～E7 を定義する）

E6.他の機能によって決定される当該事象の開始／終了条件

E7.当該事象の実現方法

（装置制御のための信号操作手順、および、実現に関わる諸パラメータの値）

[例 4.2]（装置モデルの例）

```
E0:搬入コンベア
E1:搬入コンベア
E2:搬送
E3:1つ
 case(E2＝搬送){
  E4:開始条件:P&P.Work受入.Work降下&準備->終了
     終了条件:ストッカー1.加工S.Work押出->終了 OR
              ストッカー2.加工S.Work押出->終了 OR
              ストッカー3.加工S.Work押出->終了 OR
              ストッカー4.加工S.Work押出->終了
  E5:Work降下&準備→Work移送→Work押出
   case(E5＝Work降下&準備){
    E6:開始条件:－
       終了条件:－
    E7:(P&P.putWork,P&P.advance)
   }
   case(E5＝Work移送){
    E6:開始条件:－
       終了条件:－
    E7:watchEndLine(),(advance(),stop())*
   }
   case(E5＝Work押出){
    E6:開始条件:－
       終了条件:－
    E7(stocker)
   }
 }
```

```
E0:ストッカー0(ストッカー2,3,4も同様)
E1:ストッカー1（ストッカー2,3,4)
E2:加工S
E3:1つ
 case(E2＝加工S){
  E4:開始条件:搬入コンベア.搬送処理.Work移送->終了
     終了条件:－
  E5:Work押出→Work加工→Work払出
   case(E5＝Work押出){
    E6:開始条件:－
       終了条件:－
    E7:pushWork(1)
   }
   case(E5＝Work加工){
    E6:開始条件:－
       終了条件:－
    E7:
   }
   case(E5＝Work払出){
    E6:開始条件:－
       終了条件:－
    E7:dropWork(1)
 }
```

プロセスモデル

プラントで製造する製品の製造手順を示したものを「製造レシピ」と呼ぶ。プロセスモデルとは、このような製造レシピの情報を定義するためのモデルである。製造レシピは工程（製品の生産・加工を行うための作業の手順、または、段階）の半順序関係として定義される。各工程は当該工程が属するレシピ名、工程ごとに独立に与えられる工程番号を持ち、以下に示すような形式で記述させる。

「レシピ名と工程番号：(着目機能名:X)により、(着目機能名:Y)から、(着目機能名:Y)へ、(処理)をする」或いは、

「レシピ名と工程番号：(着目機能名:X)により、（着目機能名:Y)へ、(処理)をする」

また、工程間の半順序構造を表現するためには、リンク構造を採用する。具体的には、当該工程の次に実行させる工程の番号を付記することにより、リンクをはり、一連のレシピの流れを表現する。なお、以上のような半順序構造は、後に、プロセスネットワーク型モデル上にアークの向きとしてマッピングされる。

[例 4.3] (プロセスモデルの例)

L1(2): パーツフィーダ上の Work を終端まで移送する

L2(3): Ｐ＆Ｐのアームでパーツフィーダ上の Work をつかみ上げる。

L3(4) : 搬入コンベア上まで移動する。

L4(5):搬入コンベア上に Work を置き、ピック＆プレースのアームをパーツフィーダ上へ移動する。(次の Work に備えるため)

L5(6): 搬入コンベア上の Work をストッカー1 の位置まで移送する。

L6(7): 押出シリンダー1～4 のいずれかの位置まで Work を移送し、それと対のストッカーへ押し出す

L7(8): 押し出されたストッカーにて Work を加工する。

L8(): 加工が終了したストッカーのコンベアで Work を系外へ払い出す。

これと同じ内容を、上述の記述形式に従って書き直すと以下のようになる。

L1(2).パーツフィーダにより、　　処理対象を左端まで運ぶ。

L2(3).Ｐ＆Ｐにより、　　パーツフィーダから、処理対象を取る。

L3(4).Ｐ＆Ｐにより、　　搬入コンベアの左端へ処理対象を運ぶ。

L4(5).Ｐ＆Ｐにより、　　搬入コンベアへ、　　処理対処を置く。

L5(6).搬入コンベアにより、　　ストッカー1 の前まで処理対象を運ぶ。

L6(7). 搬入コンベアにより、ストッカー1,2,3,4 いずれか一つへ、
処理対象を運んで押出しシリンダーで押出す。
(以降、ここで選ばれたストッカーをＳ*と書く。)

L7(8).Ｓ*により、　　処理対象を所定の時間処理する。

L8(　).Ｓ*により、　　処理対象を系外に払い出す。

不確定環境モデル

製品の加工やコンベアによる移動などに関わる所要時間は常に確定的ではなく、ある程度の時間幅で変動する。また、生産要求の発生間隔や装置の故障などのような不確定な出来事も発生する。制御系を設計する上ではこのような不確定性も無視できず、その特徴を抽出し、何らかの形で構築するモデルの中に取り込まなければならない。不確定環境モデルとは、以上のような不確定性に関する情報を定義するためのモデルである。

[例 4.4] (不確定環境モデルの例)

(例１)

内容：生産要求の到着間隔

分布型：一様分布　(分布の範囲は 10 以上 30 以下の整数)

平均：20

(例２)

内容：加工 X の遅延時間

分布型：指数分布 $f(x) = \lambda e^{-\lambda x}$

平均値，分散：$\lambda = 0.02$

運用戦略モデル

運用戦略モデルとは、以上のような各種モデルでの取り決めの後もなおシステム上存在する、操作上の種々の自由度に対し、何らかの取り決めを与えるためのモデルである。具体的には、サブシステム間での通信のタイミング、処理装置の選択の自由度に対する制御ルール(使用順序や優先順位)、装置使用の競合解消のための制御ルール、各設備に関する操作上の設定値、等を定義するためのモデルである。

[例 4.5] (運用戦略モデルの例)

(例１)

処理 X のための加工装置(T1～T4)の使用優先順序は、T1，T2，T3，T4 とし、空いているストッカーの中で最も優先度の高いものを使用するものとする。

(例２)

パーツフィーダの移動速度：２秒

ロボットアームの開閉速度：０．２秒

２．ユーザ仕様記述からプロセスネットワーク型モデルの生成方法について

Fig.4.5 は、本稿で提案する「プロセスネットワーク型モデルの生成の流れ」を概念的に説明する図である。挙動単位と一対一対応付けてノードを生成し、異なる挙動単位間の関係をストリーム通信を表すアークで表現するというのが基本的な考え方である。

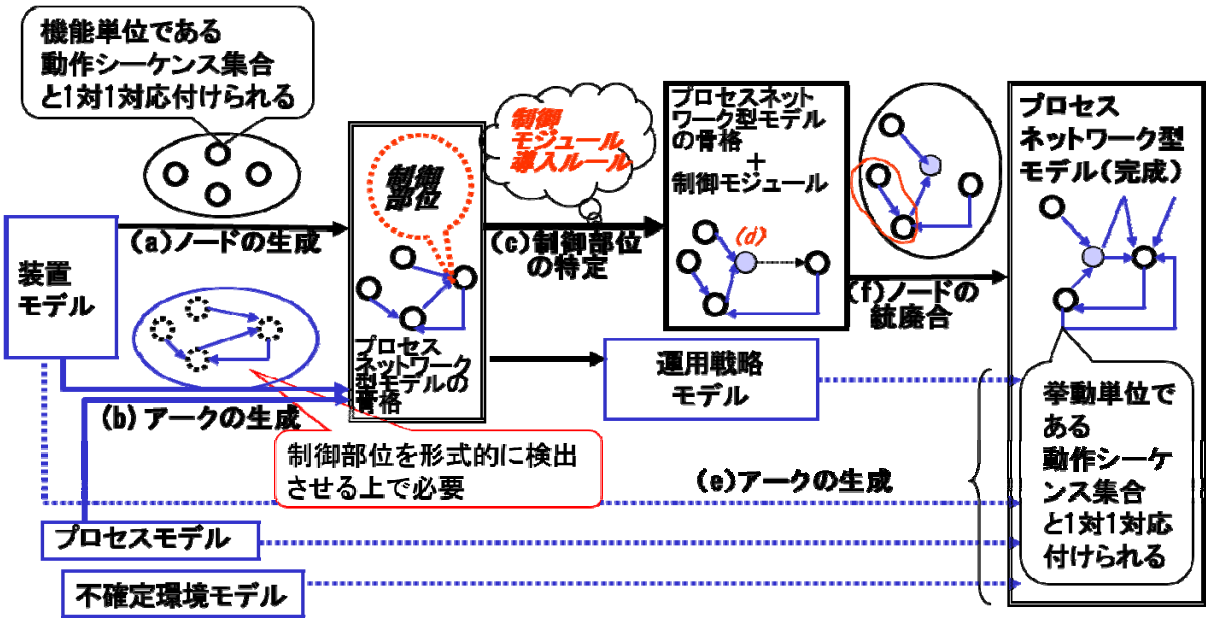


Fig.4.5：　プロセスネットワーク型モデルの生成の流れ

基本的な流れは以下の通りとなる。（説明中の(a)～(f)は、Fig.4.5 中の(a)～(f)に対応付けられる。）

Step1. プロセスネットワーク型モデルの骨格を生成

（a）ユーザ仕様記述で定義された機能単位と 1 対 1 対応付けてノードを生成する。（ノードと 1 対 1 対応付けて動作マネージャを導入する。）

（b）制御を要する箇所を特定させる上で必要なアークを生成する。

Step2. 制御モジュールの導入

(c) Step 1 で生成したプロセスネットワークの骨格に“制御モジュール導入ルール”(後述)を適用し、制御部位を特定する。

(d) 各々の制御モジュールをノード形式で導入する。

Step3. プロセスネットワーク型モデルの完成

(e) 残りのアークの追加導入を行なう。

(f) 実行効率を上げるためのノードの統廃合を行なうことによりプロセスネットワーク型モデルを完成させる。ここでのアークは、他のノードをアクティブにするトリガとなるイベントの通信路となる

上記の流れを、Fig.4.6 を使って更に補足説明する。Step1-(a)は、要求された処理全体（ユーザ仕様記述）について、頻繁な排他制御を必要とする処理同士は一つのノードにまとめるという処理を表している(Fig.4.6-1)。こうすることにより、そのような処理は、同じ動作マネージャによって一元管理することができる。このように「互いに頻繁な排他制御が必要な処理の集まり」であるノードの内部では、各処理は互いに XOR の関係にある(つまり、同時実行できない)イベント列の集まりとして表現する。更に、Step1-(b)は、頻繁ではないが、ノードを跨いで何らかの制御を必要とする処理同士に対し、制御のためのメッセージの授受を表すアークを導入するという処理を表している(Fig.4.6-2)。具体的には、メッセージを送る側のノードから出て受け取る側のノードへ入るアークを導入するという処理である。この時、アークの根元及び先端は、より詳細に、ノード内部のイベント列上の適切な位置に与える。具体的には、例えば、当該メッセージを出すタイミングは「イベント X が生起した直後」、或いは「イベント X を生起させる直前」のように定める。また、Step1-(b)にある「制御を要する箇所を特定させる上で必要なアーク」というのは、特定のノードをアクティブにするための信号の授受を行うアークという意味である。図的には、アークの先端がノードの枠上に達するようなアークのことであり、詳細は後述する。

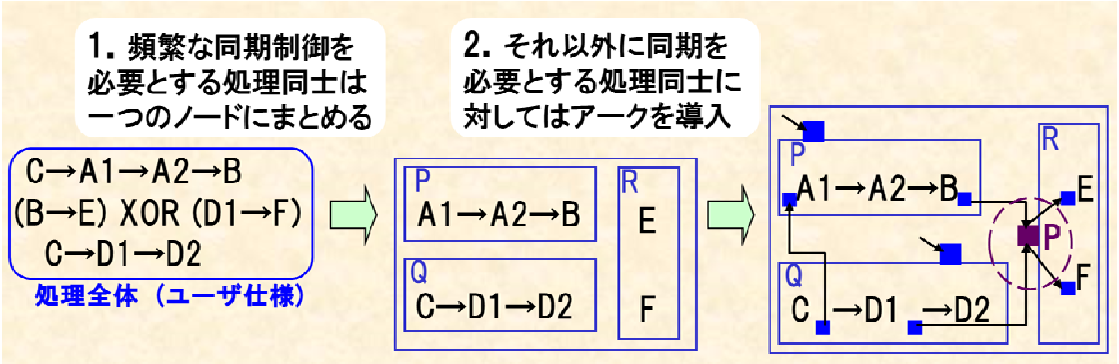


Fig.4.6:　プロセスネットワーク型モデル作成の流れ

以下では、「制御モジュール導入ルール」について説明した後、まず、プロセスネットワーク型モデルの表現方法を定義する。具体的には、提案するプロセスネットワーク型モデルの生成の流れを、形式的手順により実現可能とするような、プロセスネットワーク型モデルの表現形式を定義する。更に、ミニ FA プラントの例を使って、プロセスネットワーク型モデル生成の全体の流れについて、具体的に示す。

制御モジュール導入ルール

「制御モジュール導入ルール」とは、何らかの制御を与えるべき箇所と与えるべき制御の型を、プロセスネットワーク型モデルの構造的特徴から識別させるためのルールである。ここでの「制御の型」とは、4.2.2 節述べた“Merge 型”と“Dispatch 型”の 2 種類となる。

・Merger 型： 異なるノードから出るアークの先端が複数集まる箇所に与えるべき制御モジュールの型（装置使用の競合に対する制御モジュール）

・Dispatch 型 : 同一ノードから出る複数のアークが、各々異なるノードへ入る構造を持つとき、当該複数アークの根元が集まる箇所へ与えるべき制御モジュールの型 （装置選択の自由度に対する制御モジュール）

プロセスネットワーク型モデルの表現方法の定義

Fig.4.7 は、本稿で定義する表現形式で記述したプロセスネットワーク型モデルの例である。以下、この例を使って各記述形式を説明する。

・プロセスネットワーク型モデルは、図的にはノードは四角の枠で表し、アークは矢印で表す。

（例）ノード : N1，N2　　アーク : A1

また、ノードとアーク以外の要素は以下のような取り決めに従って生成する。

・ノードの内部プロセス： 挙動単位が含む動作シーケンスの各々と一対一対応付けて生成する。

（例）動作シーケンス： L1, L2，L3,･･･，L4,･･･

内部プロセス： {L1,L2}，{L3,･･･},{L4,･･･}

挙動単位： {{L1, L2}}，{{L3,･･･},{L4,･･･}}

・内部プロセスの構成要素： 事象と一対一対応づけて生成する。

事象： L1，L2，L3，･･･，L4，･･･

・ノードインターフェース： アークの接続口（接続元／接続先）と対応付けて生成する。図的には、

ノードを表す枠上、或いはノード内部の黒点として表す。

ノードインターフェース： *p1，p2，p3*

p1 : *p1* がアクティブとなった直後、事象 L1 の実行は開始される。

p2 : 事象 L1 の実行終了直後にアクティブとなる。

p3 : *p3* から信号が入った直後、ノード N2 がアクティブとなる。

アーク A1 の接続元 *p2* : *p2* がアクティブになった直後、*p2* から信号が発信され A1 を介して *p3* へ送られる。

アーク A1 の接続先 *p3* : *p2* からの信号が到着した直後、*p3* がアクティブとなり、ノード N2 がアクティブと成り、内部プロセスの何れか一つがアクティブとなる。

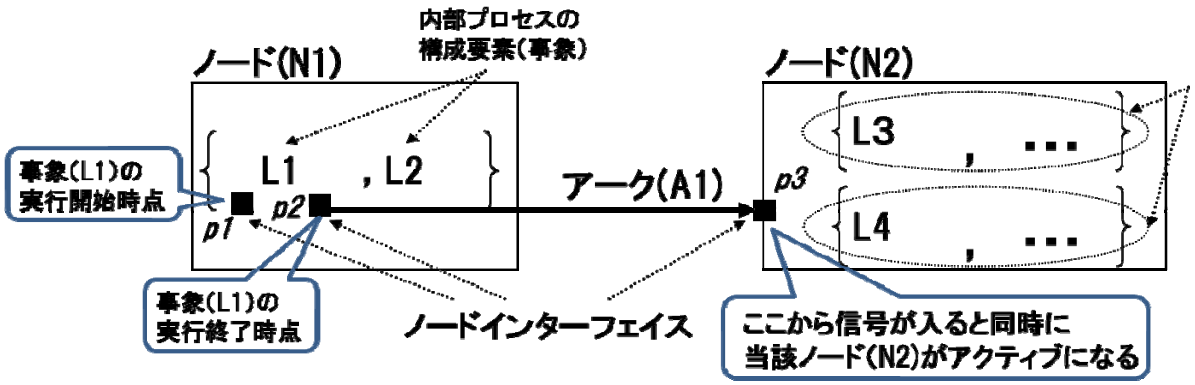


Fig.4.7:　プロセスネットワークの構成要素の図的表現

先に述べた、「モデルの骨格を生成する際に導入するアーク」とは、上記 *p3* のように、ノードを表す四角枠の側上のノードインタフェースに入るアークである。つまり、このようなアークが、「ノードをアクティブにする信号の授受を行わせるアーク」である。

[例 4.6] （制御モジュールの導入の例）

制御モジュール導入ルールより、左図の点 p は、Merge 型の制御モジュール導入箇所である。その箇所(点 p)に Merge 型の制御モジュールを導入すると、右図のようなプロセスネットワーク型モデルが得られる。

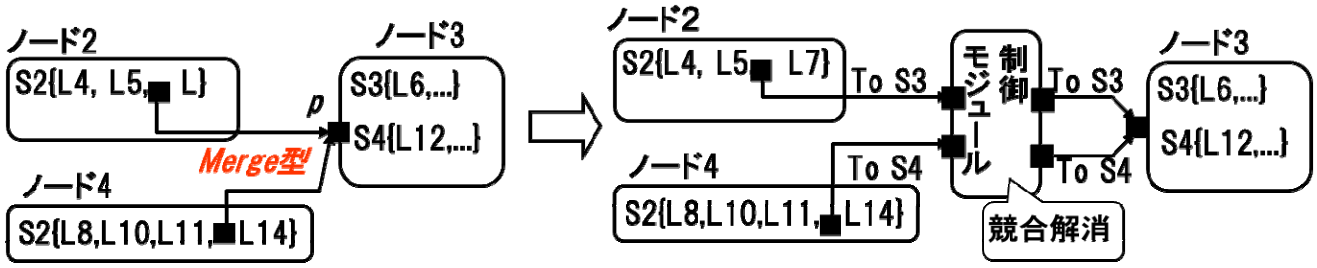


Fig.4.8: 制御モジュールの導入

[例 4.7] （ノードの統廃合の例）

ノード 2 の振る舞いは、ノード 1 の振る舞いの内部に完全に埋め込まれる。言い換えると、2 つのノード内部の動作シーケンスは、ノード 1 の事象 L4 で始まり同ノードの L7 で終わる非決定性のない 1 本の事象列の形で書ける。

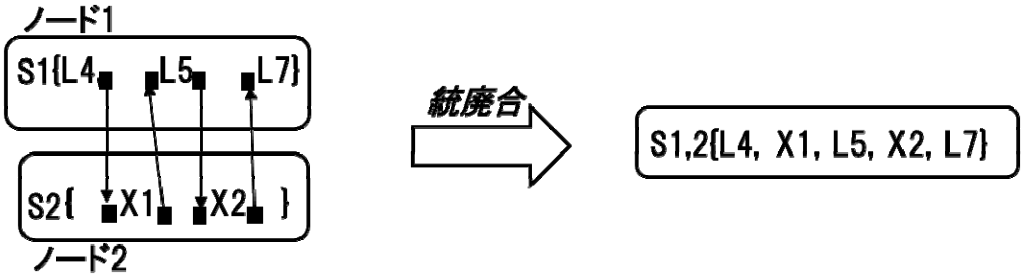


Fig.4.9: ノードの統廃合

ミニ FA プラントのプロセスネットワーク型モデルの生成

以上述べてきたように、プロセスネットワーク型モデルの生成は、（1）骨格部分の生成（ノードの導入、骨格部分のアークの生成)、（2）制御モジュールの導入、（4）骨格以外の生成、の順に行う。以下、各々の方法を、ミニ FA プラントの具体例を使って順に示す。

（1）骨格部分の生成

Fig.4.10 は、4.3.1-1 で定義したフォーマットに従って記述されたユーザ仕様記述から、プロセスネットワーク型モデルの骨格への、データの変換のためのマッピングの一部を表している。

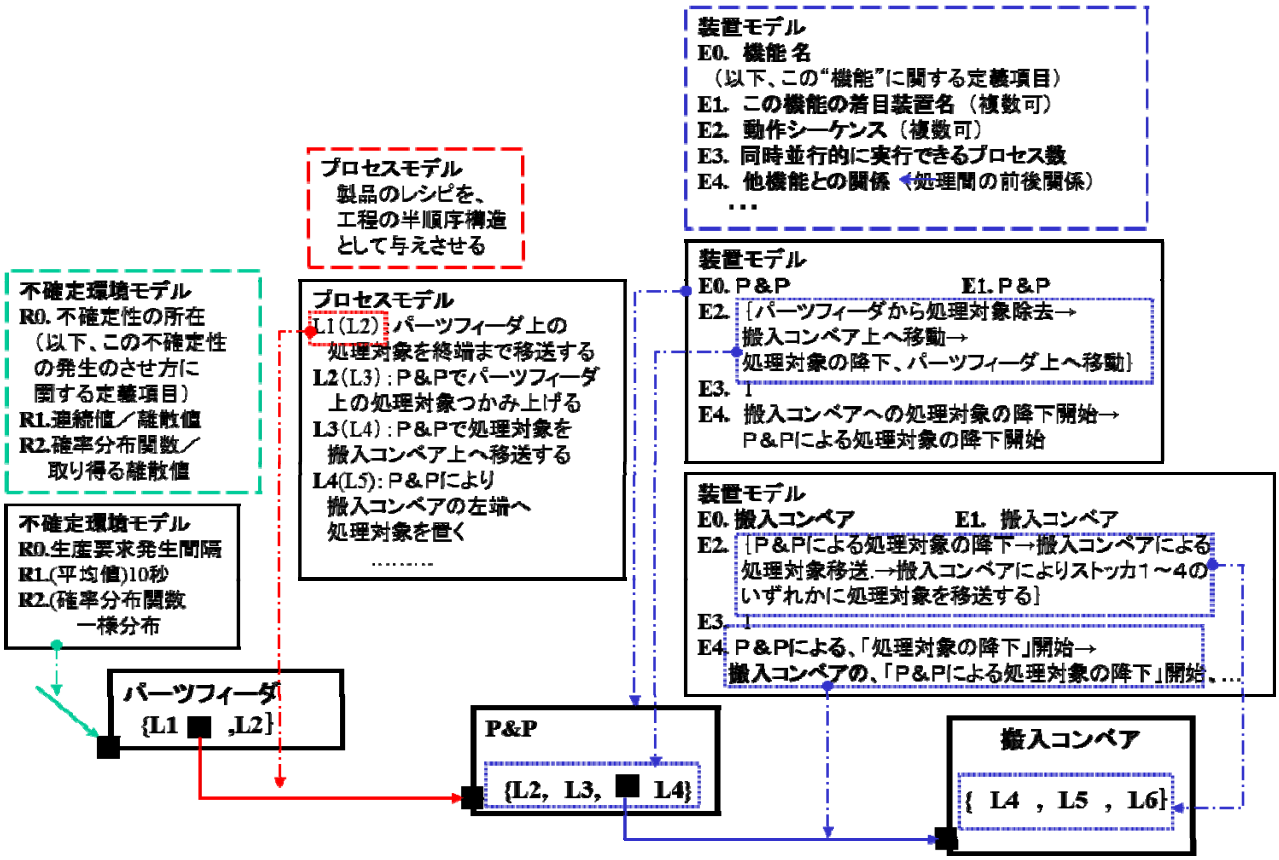


Fig.4.10:　プロセスネットワークの骨格の生成

（2）骨格部分全体と制御モジュールの導入

Fig.4.11-(A)は、得られたプロセスネットワーク型モデルの骨格全体を表す。同図の(B)は、(A)に制御モジュール導入ルールを適用して特定された箇所に、Dispatch 型の制御モジュールを導入したものである。

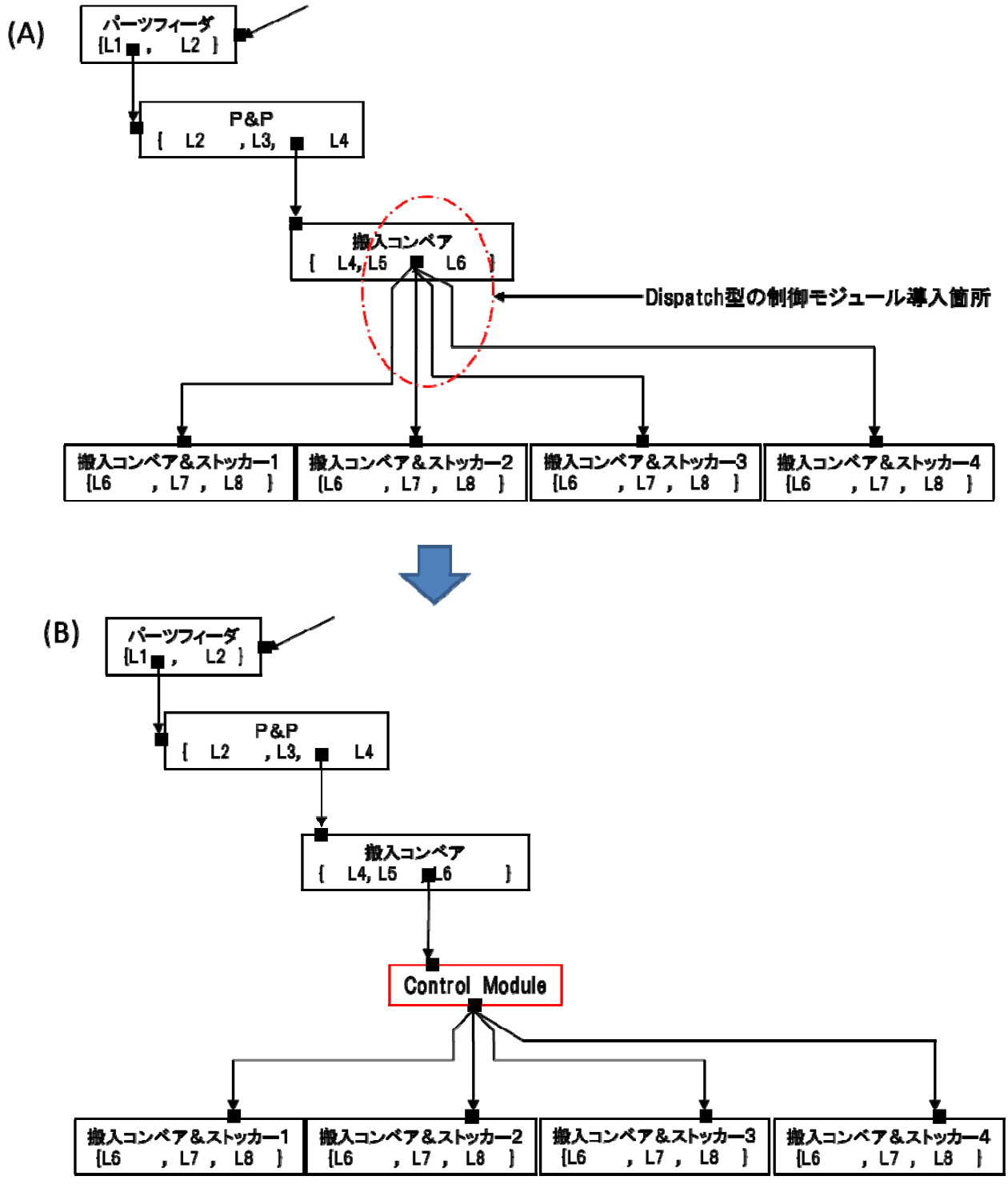


Fig.4.11:　プロセスネットワークの骨格と制御モジュールの導入

（３）骨格以外の部分の生成（アークの追加導入）

Fig.4.12 は、4.3.1-1 で定義したフォーマットに従って記述されたユーザ仕様記述から、プロセスネットワーク型モデルへの、骨格以外の部分の変換のためのマッピングの一部を表している。

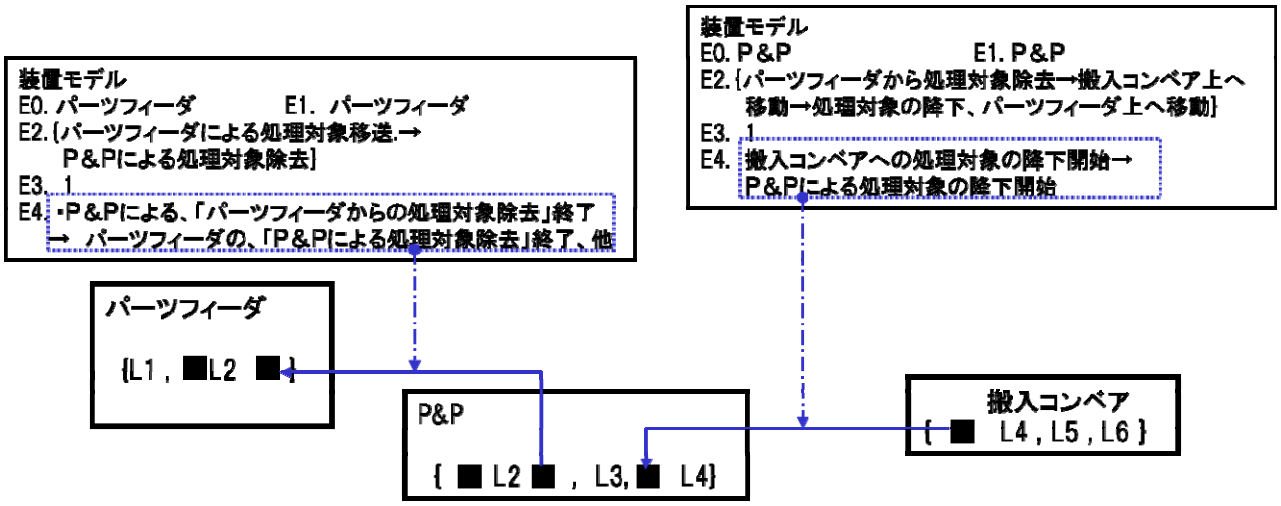


Fig.4.12:　プロセスネットワークの骨格以外の生成

（4）プロセスネットワーク型モデル（全体）

Fig.4.13 は得られたプロセスネットワーク型モデルの全体である。

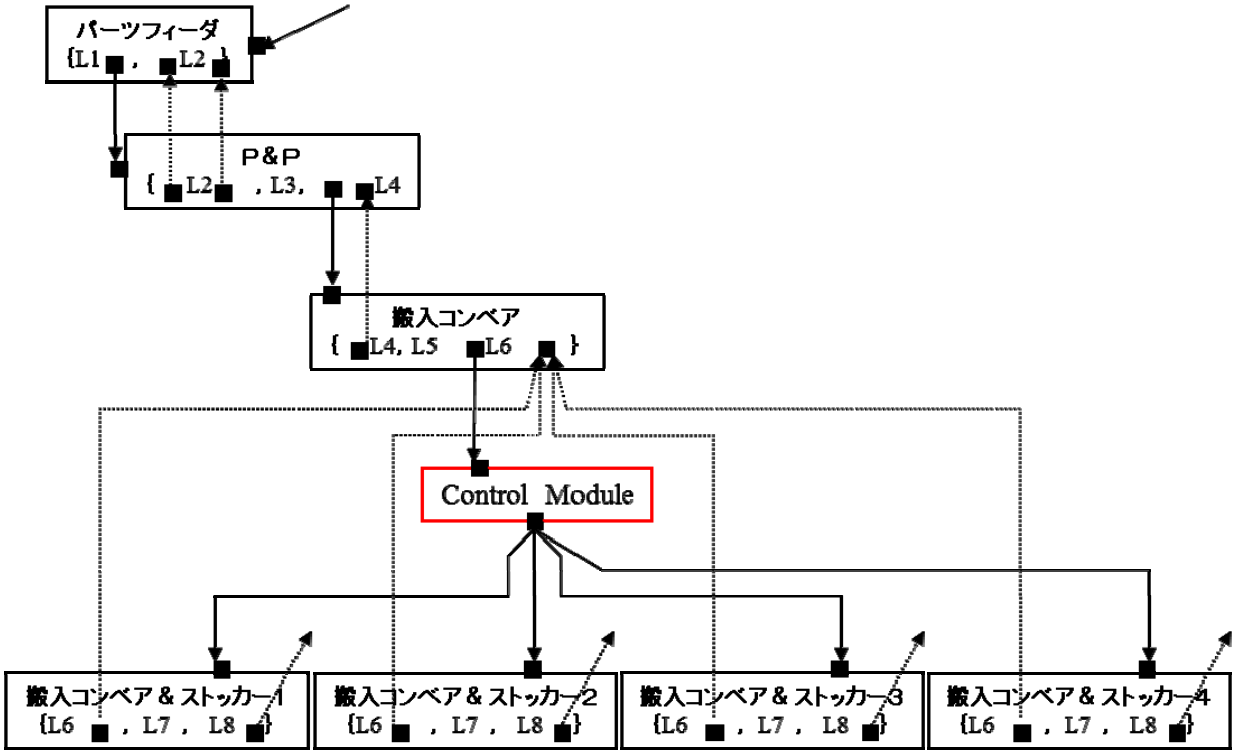


Fig.4.13 プロセスネットワーク型モデル全体

4.3.2 プロセスネットワーク型モデルから拡張時間ステートチャートへ変換する方法について [28)]

本節では、まず、拡張時間ステートチャート(ETSC)上の記述と実機動作との対応関係を付け易くするための記述上の制限を与える（Restricted ETSC の導入)。次に、プロセスネットワーク型モデルから ETSC への変換方法について示す。

Restricted ETSC の導入

プロセスネットワーク型モデルから ETSC への変換を行う際に、生産システムの制御系の開発をより体系的に行い易くするために、次のような 4 つの記述上の制限を与える。これは、制御対象と制御系動作との関係付けを容易に行えるようにするための制限であり、こうして作られる ETSC のサブクラスを特に“Restricted ETSC”と呼ぶ。（Fig.4.14 参照）

（１）「制御対象全体の動作」は、常に“ETSC 全体”として記述すること　(Fig.4.14-(1))

（２）「一まとまりの装置構造の動作」は、常に“一つのオートマトン”として記述 すること(Fig.4.14-(2))

（３）「一工程の実行を遂行するための動作」は、常に“一つの遷移のアークの内部情報”として記述すること　(Fig.4.14-(3))

（４）「プラントの安全性を維持するための動作」は、常に“一つのロケーションの内部情報”として記述 すること(Fig.4.14-(4))

ここで（３）,（４）の“内部情報”とは、例えば上記（３）のケース(Fig.4.14-(3))の場合は、「当該アーク t1 の遷移が生起した際に実行されなくてはならない部分操作」に関する一連の情報全体を指す。

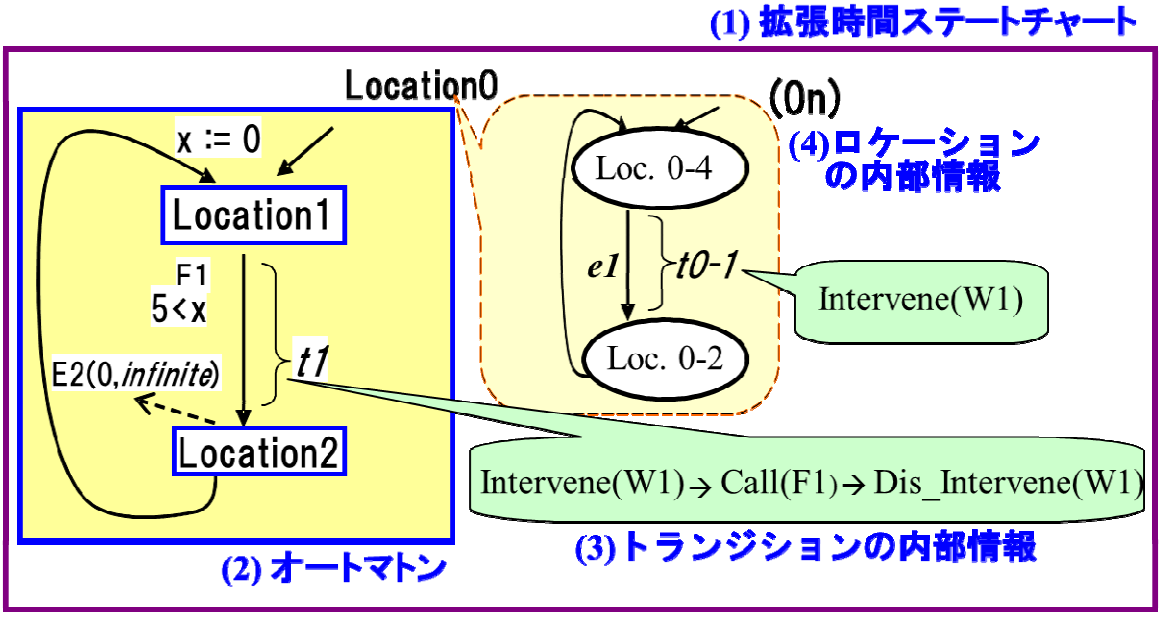


Fig.4.14:　挙動表現モデルとしての拡張時間ステートチャート

プロセスネットワーク型モデルから拡張時間ステートチャートへの変換方法

まず、プロセスネットワーク型モデルの各ノードを ETSC の独立な並行プロセスである時間オートマトンに各々対応付ける(Fig.4.15-(1))。そして、ノード内部の動作シーケンスは、メッセージの授受を行うタイミングとして指定された箇所で各々区切り(Fig.4.15-(2))、区切られた各区間(部分シーケンス)を当該時間オートマトンのロケーションとして各々対応付ける(Fig.4.15-(3))。ここで、動作シーケンスを以上のような形で区切る理由は、プロセスネットワーク型モデルのアークの根元(メッセージ出力元)と先端(出力先)を、ETSC 上でのイベントの生成箇所(遷移のアークの先端)と発火箇所(アーク上)となるように対応付ける上で、ロケーション間の境界が必要だからである。ただし、ロケーションの数は必要最小限に抑えることがモデルの簡潔な記述上望ましい。そのため、本稿では、動作シーケンスを必要最低

限の箇所で区切るためのルールを検討し定めた（後述）。また、ETSC の並行性としては、プロセスネットワークのノード間の並行性をETSCの並行性としてそのまま継承させる。また、ETSCの階層性も、プロセスネットワーク型モデルの 3 階層(即ち、全体,ノード，イベント列の各区間)をそのまま継承し、ETSC 上では、全体、時間オートマトン、ロケーションとして表現する。また、各制御部位には、スケジューリング結果から抽出される順序情報を各々与える。以上をまとめると、プロセスネットワーク型モデル(PNM)から拡張時間ステートチャート(ETSC)への変換のための対応関係は以下の通りである。

PNM から ETSC への変換のための対応関係

・PNM 全体　→　ETSC 全体

・ノード　→　時間オートマトン

・ノード内部のプロセス　→　時間オートマトンを構成するシーケンシャルなパス

・内部プロセスの部分シーケンス　→　時間オートマトンのロケーション　（※）

・アーク(を流れる信号)　→　イベント

・アークの根元　→　生成イベント

・アークの先端　→　発火イベント

以上のような変換により、プロセスネットワーク型モデルから拡張時間ステートチャートが得られる。

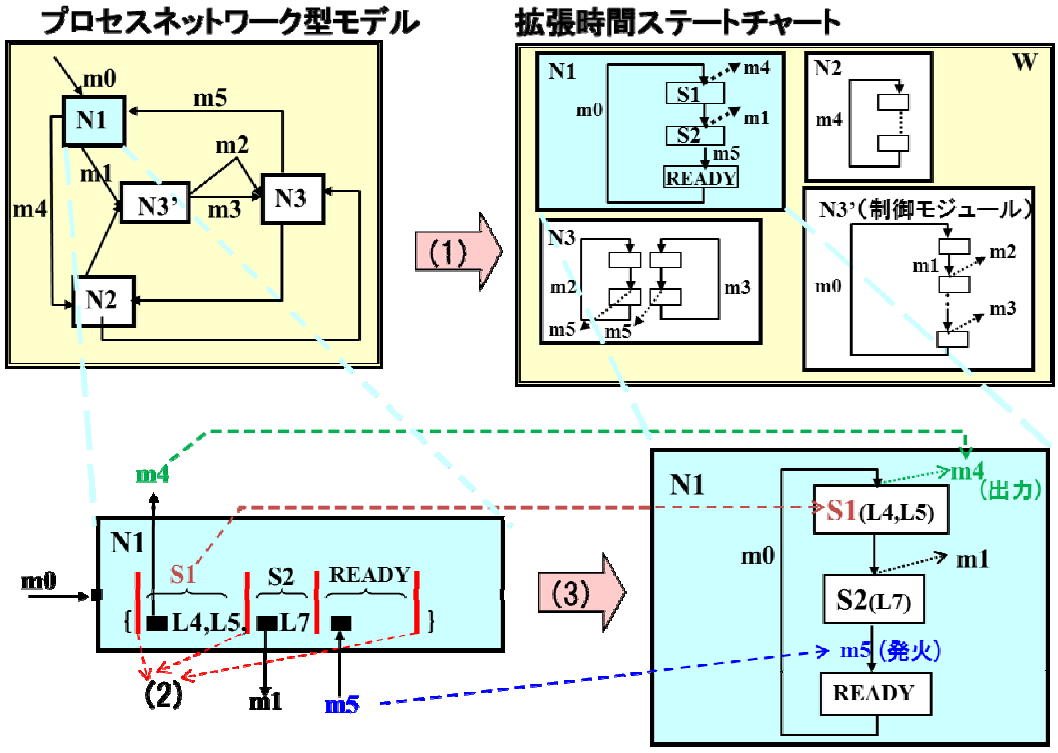


Fig.4.15:　プロセスネットワーク型モデルから拡張時間ステートチャートへの等価変換

[例 4.8] （プロセスネットワークからオートマトンへの変換の例）

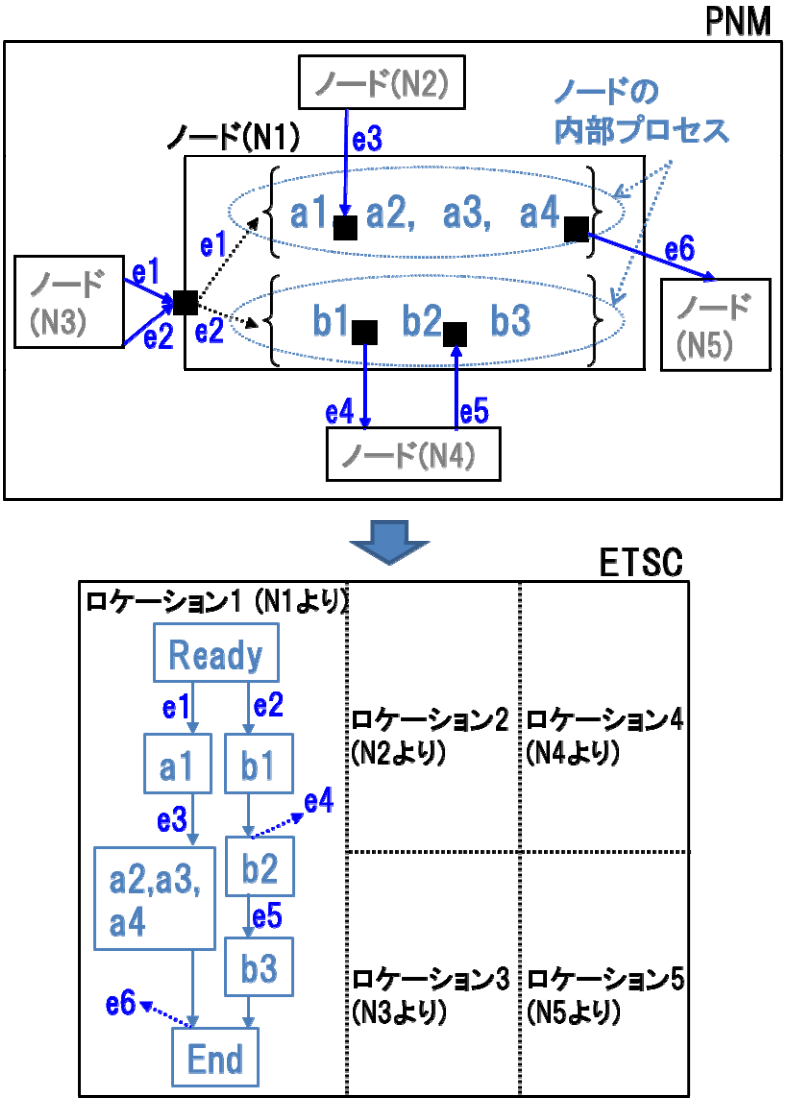


Fig.4.16: プロセスネットワークからオートマトンへの変換

ETSC のロケーションを必要最低限導入するためのルールについて

（-PNM の動作シーケンスの区切り目を必要最低限とするためのルール-）

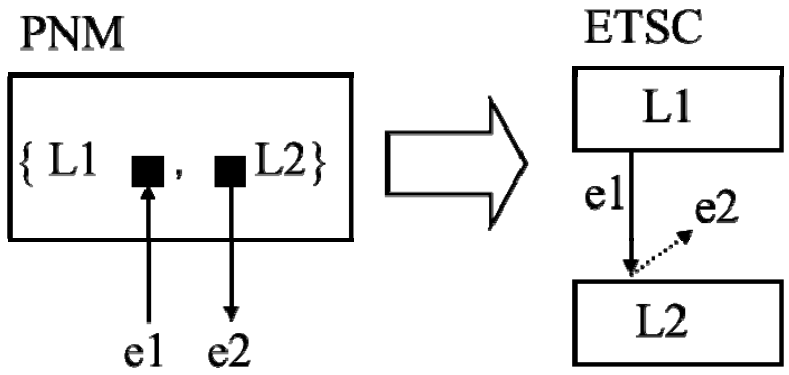
上述したとおり、本稿では、PNM からの変換先の ETSC のロケーション間の境界が必要最低限となるようにするため、その境界の箇所を決める変換もとの PNM の動作シーケンスの区切り目を必要最低限とするようなルールを定めた。

ルールを決める上での基本的な考え方は次の通りである。まず、ETSC 上、ロケーション間の境界が必要な箇所は、イベントの発火、及び、出力の箇所である。しかし、このようなイベントの発火，出力が 2 つ以上続く場合、その 2 つのイベント間を区切る必要があるか否かは、その組み合わせによって異

なる。具体的には、1 つ目のイベントの発火/出力と 2 つ目のイベントの発火/出力が連続して生起し得るならば、これら 2 つのイベントの授受は同一の箇所から行えば良い為、区切る必要がないということである。つまり、1 つ目のイベントが何であれ、2 つ目のイベントが「出力」であれば、区切る必要はないということである。一方、2 つ目のイベントが「発火」であれば、1 つ目のイベントから連続してその発火が生起し得るかどうかは対象や状況に因って決まることである。従って、ルールとしては、「2 つ目のイベントが「発火」であれば一般には区切る必要がある。しかし、区切らなくても良いケースもある」となる。以上を踏まえ、本稿では、動作シーケンスを区切るためのルールは、以下の 4 パターンに分けて定めた。

[例 4.9] （動作シーケンスの 2 つのノードインタフェース間に区切り目が必要ないケースの例）

（1）「発火」→「出力」の場合



（2）「出力」→「出力」の場合

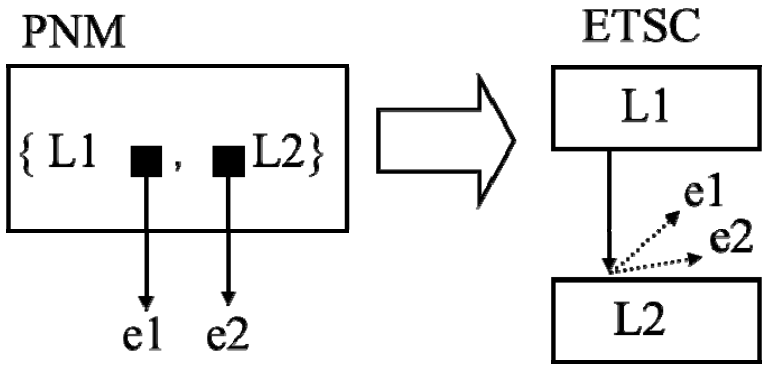


Fig.4.17:　PNM の動作シーケンスの区切り目を必要最低限とするためのルール -1

[例 4.10] （動作シーケンスの 2 つのノードインタフェース間に区切り目が必要であるケースの例）

（3）「出力」→「発火」の場合

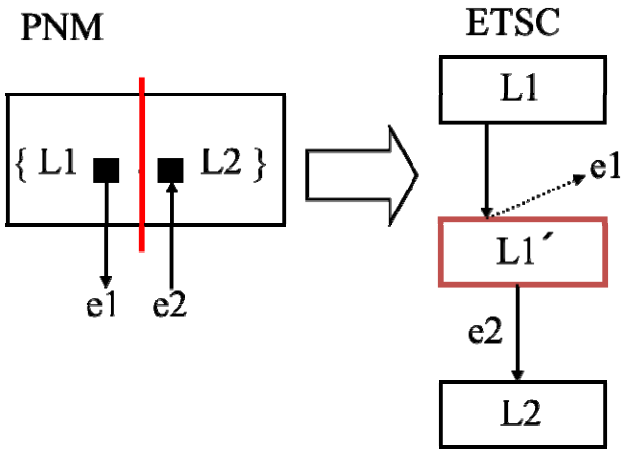


Fig.4.18： PNM の動作シーケンスの区切り目を必要最低限とするためのルール -2

（4）「発火」→「発火」の場合

イベント e1 の発火可能期間が十分長く、e2 が発火するまで、e1 の発火が他のプロセスによって妨げられることがない場合、区切る必要がない(ETSC-1)。一方、その逆で、e1 が発火可能となった時点以降 e2 が発火可能になるまでの間に e1 の発火が不能になる可能性がある場合は区切る必要がある(ETSC-2)。

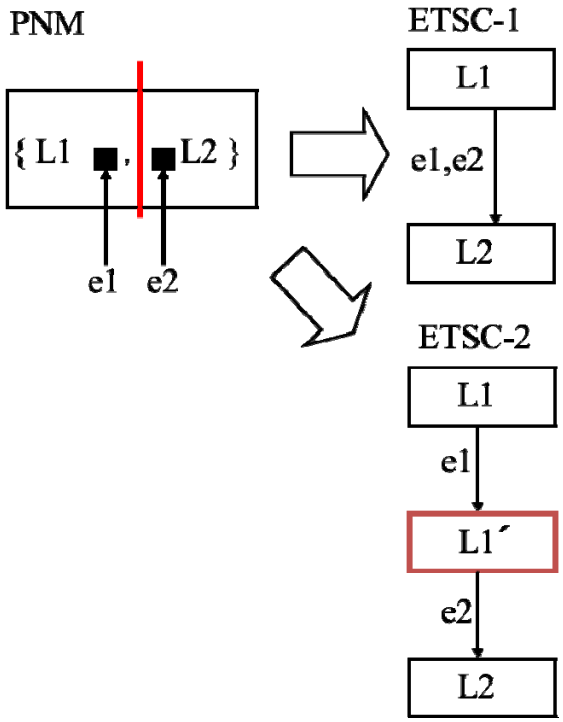


Fig.4.19: PNM の動作シーケンスの区切り目を必要最低限とするためのルール -3

4.4 R-ETSC に基づく制御系実行モジュール生成のためのテンプレートとその利用方法

4.4.1 テンプレートの導入

本提案方法では[29]、挙動モデルの動作を掌握するのに必要な最低限の情報を、プロセスの挙動を表現するための情報、イベント駆動の動作、タイマー駆動の動作、競合解消や排他制御のための動作(制御リストの埋め込み箇所とその内容)、及び、例外処理の、都合 5 種類の情報に分けて扱う。そして、この分類に基づき、ETSC から制御に必要な情報を要約・抽出するためのテンプレートとして、“プロセステンプレート”と、そこから呼び出される“イベントテンプレート”、“タイマーテンプレート”、“制御リストテンプレート ”、及び、例外処理用の“割り込みテンプレート”の、都合 5 種類を導入する(Fig.4.20 参照)。そして、制御に必要な情報を作成する際には、上記のような各テンプレートを各々必要な数複製し、それらに、挙動モデルから抽出される具体的なパラメータを与え、実際の制御系への入力となる制御系実行モジュール（以降 “実行モジュール”）の集まりを生成する。(各々を、プロセスモジュール、イベントモジュール、タイマーモジュール、制御リストモジュール(或いは、制御リスト)、及び、割り込みモジュールと呼ぶ)。

実行モジュール内部には、まず、どのテンプレートの内部にも、モジュールを実行する際の処理内容を記述するための「処理部」を持たせる。更に、プロセスモジュールをトップモジュールとした階層性に基づく実行形態を取れるようにするため、各モジュールの実行開始タイミングは次のように定める。即ち、プロセステンプレートでは、そのプロセスの実行開始条件を、アクティブであるべきロケーション、及び、発火可能であるべきイベントの組として、任意に指定できるようにする。また、それ以外の 4 種類のテンプレートに関しては、自身以外の何らかのモジュールに呼び出される時を実行開始タイミングとする。つまり、どのモジュールを何時呼び出すかは、各モジュールの処理部で指定する。

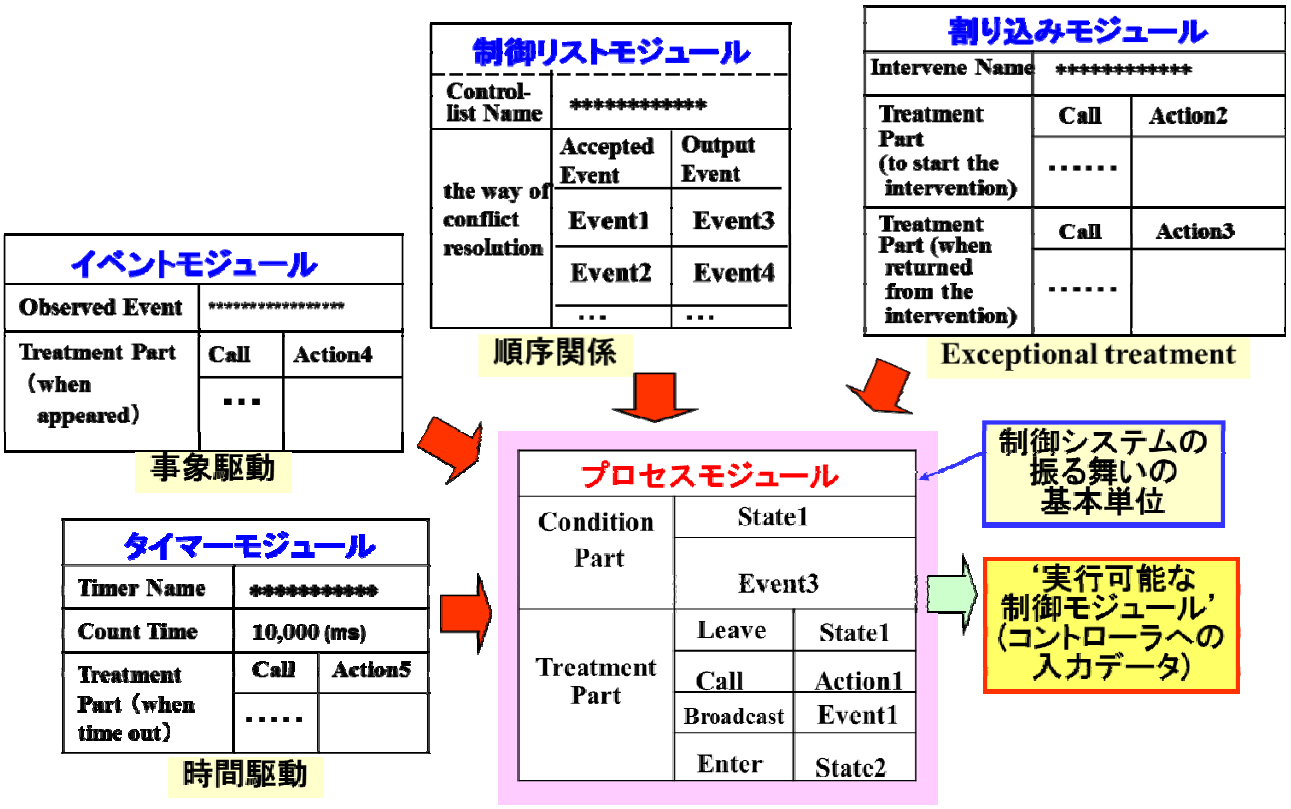


Fig.4.20:　制御系実行モジュール生成のための 5 種類のテンプレート

4.4.2　テンプレートの利用

制御系への入力データの R-ETSC からの抽出は、(1)各テンプレートのインスタンスの生成、(2) 各インスタンスの内部データの生成、の 2 段階で行う。

(1)のインスタンスの生成では、プロセステンプレートは R-ETSC の遷移のアークと 1 対 1 に対応付けて生成し、その他のテンプレートは、何れかのモジュールの実行中に呼び出される可能性のあるもの全てに関し生成する。

(2)の内部データ生成では、プロセステンプレートに埋め込む情報としては、実行開始条件(アクティブであるべきロケーションとイベントの組)と処理内容(トークンの移動，イベントの生成，各種モジュール呼び出し)を、R-ETSC モデルから抽出し、その他のテンプレートに関しては処理内容のみをそれぞれ抽出する。また、抽出した処理内容は、処理すべき順番に処理部に埋め込む。例として、プロセスモジュールの生成方法を示す。(Fig.4.21 参照)

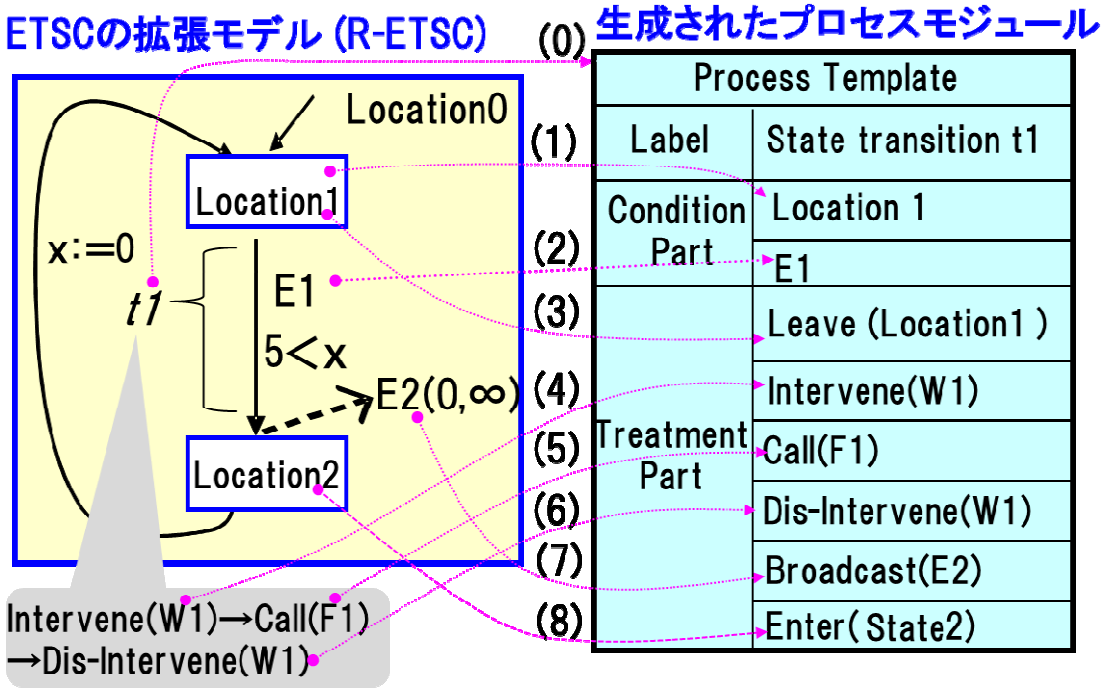


Fig.4.21:　テンプレートを用いた制御系実行モジュールの生成

プロセスモジュール生成の流れ

(1)　まず、R-ETSC の一つの遷移「Location1→Location2」に対し、プロセステンプレートを 1 つ導入する。(Fig.4.21-(0))

(2)　次に、当該遷移の遷移条件と、遷移に伴って実行すべき内容を各々抽出し、それぞれ「条件部」と「処理部」へ挿入する。この例の場合、まず Location -1 がアクティブであり(Fig.4.21-(1))、かつ、イベント“E1”が発火可能である(Fig.4.21-(2))、という 2 種類の遷移条件を抽出し、テンプレートの条件部にそれぞれ挿入する。また処理部には、6 種類の処理内容を抽出し、処理すべき順番に挿入する。即ち、「Location 1 を OFF にする」(Fig.4.21-(3))、「装置 W1 へ割込みをする」(Fig.4.21- (4))、「動作関数 F1 を実行する」(Fig.4.21- (5))、「W1 への割込みを解除する」(Fig.4.21- (6))、「イベント E2 を生成する」(Fig.4.21- (7))、「Location2 を ON にする」(Fig.4.21- (8))、という 6 つの処理内容を抽出し、テンプレートの処理部に順番に挿入する。

プロセスモジュール以外のモジュールの生成の流れ-1（必要なテンプレートの生成）

例えば、Fig.4.21 の例で言えば、プロセスモジュール生成中に抽出された情報の中に、「Intervene(W1)」,「Dis-Intervene(W1)」という割り込みに関する命令が存在する。このように、割

り込みに関する命令の抽出が行われたとき、具体的には、“Intervene”,或いは“Dis-Intervene”という特定の文字列で始まる命令を抽出したとき、割り込みテンプレートの導入を行う。同様に、タイマーテンプレートの導入は、文字列“Reset”，“Suspend”,“Resume”で始まる命令、及び、タイマー名を変数とする時間不等式を抽出したときに、テンプレートの導入を行う。また、イベントテンプレートは、“ObservedEvent”で始まる命令を抽出した直後に、テンプレートの導入を行う。

以上のように、必要なテンプレートの導入は、命令を表す文字列の検出を契機に、機械的に行うことが出来る。ただし、命令とテンプレートの関係は 1 対 1 ではなく多対 1 である。例えば、一つの装置に対する“Intervene”で始まる命令と“Dis-Intervene”で始まる命令は 1 セットで一つのテンプレートに埋め込む情報として扱われなくては成らない。同様に、一つのタイマーに関する複数の命令(リセット、停止、時間不等式等)に対しては一つのテンプレートを、一つの観測イベント（センサー情報）に関する複数の命令に対しては一つのテンプレートを、重複なく必要最低限に導入する必要がある。このような重複を防ぎながらテンプレートを導入することは、命令の後のカッコの中にかかれた装置名、タイマー名、イベント名を併せて参照することで、容易に行える。以上のことから、与えられた ETSC が持つ情報を重複なく網羅的に取り出すためのテンプレートの生成は、完全に機械的に行うことが出来る。

プロセスモジュール以外のモジュールの生成の流れ-2（テンプレートに埋め込むための情報の抽出）

例えば、割り込み処理に関して言えば、どの装置を、どのタイミングで一旦止める必要があるのか、或いは、一旦止めた装置の動作を再開してよいのは何時なのか、といったことは、プラントの持つ物理的な制約で決まる事柄である。従って、この種の情報は、制御系設計のフレームワークの最上流のユーザ仕様記述で、与えられる必要がある。以上のことから、割り込みテンプレートへの情報の埋め込みでは、まず、当該テンプレートに対応付けられる装置の割り込み前後の操作方法を、ユーザ仕様記述の中から見出し、そこに書かれてある操作方法を、指定のテンプレートのフォーマットに埋め込める形に変形して埋め込むという方法で行う。また、タイマーを要する処理としては、所要時間をタイマーテンプレートへの情報の埋め込みは次のようにして行う。

4.4.3 モジュール抽出システム

R-ETSC モデルを入力データとして用いることにより、テンプレートに基づいてシステマティックな手順で実行モジュールを生成するコンピューターシステム”モジュール抽出システム”を、以上の提案手法に従い開発した。本抽出システムの入出力データの構成は以下の通りである。

＜モジュール抽出システムの入力データ＞

・R-ETSC の遷移関係を基本単位とするデータ（Transfering）:

モジュール抽出システムへの入力データの基本単位は、ETSC の「一つの遷移関係」であり、次の 6 項目の組として与える。

1. 遷移関係の識別子　2. 遷移元ロケーション名

3. 遷移先ロケーション名　4. トリガイベント名

5. 当該遷移実行時に実機へ渡すべき特定の「命令コマンド系列」を指す識別子

6. 生成イベント名

[例] PP-READY　PP-READY　PP-A2　PFA1-e　A1(※)　PPA2-s

((※) A1 は、次に説明する“命令コマンド系列”「 Call PickPlace getWork」を呼び出すための識別子)

・実機へ渡す命令コマンドの系列データ（CommandList）:

実機へ渡される命令コマンドの(一まとまりの)系列データは、他のモジュールから識別子をキーとして呼び出され、実行される。

1. 識別子

2. 他のモジュールから指定され実機に渡される命令コマンド系列 (以下“命令コマンド系列”)

[例] A6　Call AirRobot　getWork　Non-Interfere　conv1-Interference

(第 1 項目“A6”は当該系列を呼び出すための識別子を表す。第 2 項以降は、順に、アームロボットを呼び出しワークを掴め、(割り込み処理中の)コンベェア 1 の割り込みを解除せよ、となる。)

・割り込み命令に関するデータ（Interfere）:

割り込みに関する定義は、割り込み開始時にすべき処理に関する(実機への)命令(或いは“命令コマンド系列”)、及び、割り込み終了時にすべき処理に関する命令(或いは“命令コマンド系列”)の対として記述する。

1. 割り込み名（識別子）　2. 割り込み開始/終了

3. 割り込み開始/終了時に実行されるべき命令コマンド系列

[例] conv1-Interference　start　Call　Conveyer1 stop

conv1-Interference　end　Conveyer1 advance

・タイマー処理に関するデータ（Timer）:

以下の 3 項目として記述する。

1. タイマー名　　2. カウント時間　　3. タイムアウト時に実行されるべき命令コマンド系列

[例] S4BTimer　50000　Call Stocker dropWork4　Broad S4BTimer-e 1

S4BTimer (T1 はタイマー名、50000 はカウント時間、実行命令は「イベント S4BTimer-e 1 を出力せよ」となる。)

・監視すべきイベントに関するデータ（ObservedEvent）:

以下の 3 項目として記述する。

1. 当該イベントを監視すべき状況 （どのロケーションがアクティブであるときに監視すべきか)

2. 当該イベントを出力するデバイス名 （どのデバイスからの出力イベントを監視すべきか)

3. 監視イベント名

4. 当該イベントの発生確認時に実行されるべき命令 （或いは命令コマンドリスト)

[例] IC-READY　　Conveyer1　　watchEndLine

Interfere conv1-Interference Broad IC-endLine

（「ロケーション IC-READY がアクティブである間ずっと、Conveyer1 から出力されるイベント” watchEndLine “の出力を観測し、それが観測されたらならば “Interfere conv1-Interference” (割り込み処理） と “Broad IC-endLine”（イベントの出力） を実行せよ」となる。 ）

・制御方法に関するデータ（ControlList）:

①使用上の競合のある各装置に、1 対 1 対応付けて生成する。要素の並び順として、当該装置の使用許可を与える順序を与える。

②使用可能な装置に自由度のある各処理に対し、1 対 1 対応付けて生成する。要素の並び順として、当該処理のために使用させる装置の使用順序を与える。

(以下は①の場合)

1. 当該装置が使用可能となったタイミングで特定の主体(当該装置を使用していた装置等)から出されるイベント

2. 次の使用許可を特定の主体へ与えるイベント （どのデバイスへ許可を与えるか？）

3. 未実行→1，実行済み→0

[例] ICA5-e-C1　C1-ARA6-IC　1　･･･要素(1)
ICB5-e-C1　C1-ARB6-IC　1　･･･要素(2)
T2A11-e-C1　C1-ARA12-T2 1　･･･要素(3)
・・・・・・・・・・・・

（要素(1)では、「アームロボットが使用可能となったことを表すイベント“ReqFromDevice3”が出力されたならば、搬入コンベアに使用許可を与えるイベント“C1-ARA6-IC”を生成し出力せよ」となる。）。

抽出システムからの出力データ

抽出システムの出力データは各実行モジュールであり、その様式は、各種テンプレートのフォーマットの通りである。(Fig.4.22 参照)

○ **Process Module**

```
<Process>
        <Precondition> PP-READY </Precondition>
        <PreOutputEvent> PFA1-e </PreOutputEvent>
        <Command> Leave PP-READY </Command>
        <Command> Broad PPA2-s 1 </Command>
        <Command> Call PickPlace getWork   </Command>
        <Command> Enter PP-A2 </Command>
</Process>
```

○**Intervene Module**

```
<InterferenceName> conv1-Interference </InterferenceName>
        <Command-Before> Call Conveyer1 stop          </Command-Before>
        <Command-After> Call Conveyer1 advance        </Command-After>
</Interference>
```

○**Timer Module**

```
<Timer>
        <TimerName> S4BTimer </TimerName>
        <CountTime> 50000      </CountTime>
        <Command> Call Stocker dropWork4  </Command>
        <Command> Broad S4BTimer-e 1      </Command>
</Timer>
```

○**Intervene Module**

```
<Event>
        <EventListener> Conveyer1 watchEndLine </EventListener>
        <Command> Interfere conv1-Interference          </Command>
        <Command> Broad IC-endLine 1       </Command>
</Event>
```

○**Control-list-modul**

```
<ControlList>
        <ControlListName> List1 </ControlListName1>
        <ControlListElement> ICA5-e-C1 C1-ARA6-IC 1 </ControlListElement>
        <ControlListElement> ICB5-e-C1 C1-ARB6-IC 1 </ControlListElement>
        <ControlListElement> T2A11-e-C1 C1-ARA12-T2 1
        </ControlListElement>
        <ControlListElement> ICA5-e-C1 C1-ARA6-IC 1 </ControlListElement>
        <ControlListElement> T4B11-e-C1 C1-ARB12-T4 1
                ..........
</ControlList>
```

Fig.4.22:　出力データ　抽出システムによって得られた出力データ(実行モジュール)の例

4.5　計画変更にも対応可能な制御系動作メカニズムの提案

4.5.1　制御系の動作メカニズムの概観

Fig.4.23 は、本手法で設計される制御システムを駆動するためのコントローラの動作の概観を表す。具体的な処理の流れは次の通りである。

まず、各実行モジュールを次のようにして作成する。即ち、ユーザ仕様記述(Fig.4.23 - (D1))を埋め込みながらプロセスネットワーク型モデル(Fig.4.23 - (D2)) を作成し、それを ETSC(Fig.4.23 - (D3))に変換し、その ETSC からの情報抽出を各種テンプレートを用いて行うことにより実行モジュールを生成する。更に、得られた実行モジュール群を制御システムへ与え、コントローラにより実行させる(Fig.4.23 -(D4))。

制御システム内部の実行の流れは次の通りである。まず、各プロセスモジュールの条件部の真偽を現在のロケーションやイベントのＯＮ／ＯＦＦに基づきチェックする。そして、条件部の条件が全て成立しているプロセスモジュールの処理部、及び、そのようなプロセスモジュールから呼び出される実行モジュールの処理部を処理するためのスレッドを個々のモジュールごとに作成する。これらはマルチスレッド方式で並列処理させる。なお、ここで並列処理される個々の処理とは、具体的には、挙動モデル上の各ロケーションやイベントと１対１に対応付ける形で導入したロケーションやイベントのＯＮ／ＯＦＦの更新、実機の各パーツの動作(ドライバープログラム)に対応付けられる動作関数のＯＮ／ＯＦＦの更新、及び実行である。

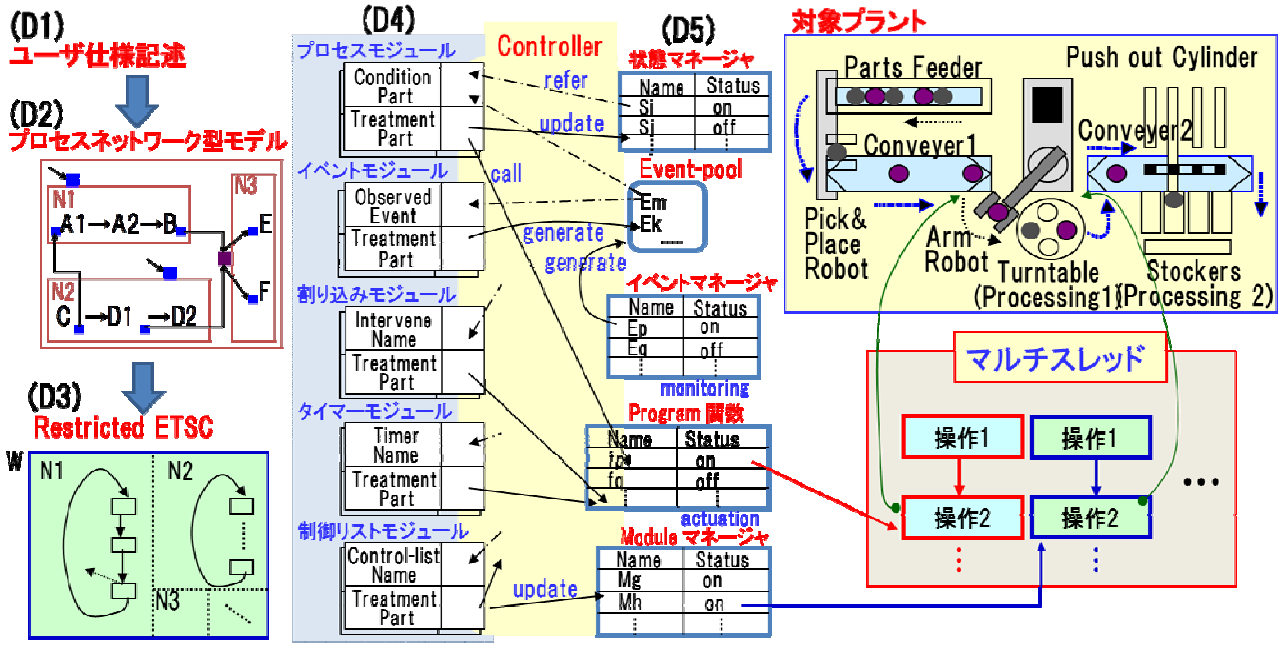


Fig.4.23:　制御システムの振る舞い

以上のように、提案する制御系動作メカニズムでは、実機の動作→制御系コントローラの内部モデル(Fig.4.23 -(D5)) → 制御系動作モジュール(Fig.4.23 -(D4))→ETSC モデル(Fig.4.23 -(D3))→プロセスネットワーク型モデル(Fig.4.23 -(D2))→要求仕様(Fig.4.23 -(D1))、というように、Fig.4.2 で示す本手法の枠組み上、隣り合うモデル同士の関係の意味を明確にすることができる。

Table4.1 は、以上のような「隣り合うモデル同士の関係」を、より詳細に、モデル上の要素間の関係として表したものである。この表は、制御系設計過程で使用するモデル間の関係を、データの変換・抽出の流れと各段階で現れる要素間の対応関係として表す対応表となっている。具体的には、まず、ユーザ仕様記述で独立に与えられる 3 種類の情報(装置特性、製造レシピ、生産環境の不確定性)は、中間モデルであるプロセスネットワーク型モデル上で統合される(4.3.1 節 - 2)。このことは、Table4.1 上では、次のように表されている。

（1）ユーザ仕様記述からプロセスネットワーク型モデルへの変換

まず、一つの装置に一つのノードが対応付けられる(図中①)。また、レシピとして与えられる「異なる装置の操作手順間の順序関係」はアークに対応付けられる(図中②)。そして、そのアークの根元の接続口であるノードインタフェース(OUT)は先行する操作を持つノードに(図中③)、同アークの先端の接続口であるノードインタフェース(IN)は後続する操作を持つノードに与えられる(図中④)。また、生産環境の不確定性に関し、生産要求の発生のタイミングや、処理の遅延の発生箇所は、各々のタイミングや箇所に相当するノード上の特定部位に、その発生を表す信号を受けるためのアーク先端の接続口であるノードインタフェース(IN)に対応付けられる(図中⑤)。なお、図中の点線矢印で示すとおり、図中③, ④, ⑤のノードインタフェースは、各装置に対して生成されるノード内部に与えられる。

以上のような変換の結果、装置特性、製造レシピ、生産環境の不確定性の 3 種類の情報は、プロセスネットワーク型モデルの各ノード上で統合される。また、運用戦略に関しては、各制御メカニズム導入箇所に対応付け、運用戦略専用のノードを導入する(図中⑤)。ただし、制御メカニズムを導入すべき箇所は、プロセスネットワーク型モデル作成段階で決まり、その後、制御方法を決定するため、変換の手順としては、他の変換の流れとは逆となる(図中⑤')。

（2）プロセスネットワーク型モデルから Restricted-ETSC への変換

まず、各装置、及び、制御モジュールに対応付けて得られた各ノードは、独立なオートマトンに各々対応付けられる(図中⑥)。ノード内部の各要素からオートマトン上の各要素への対応付けについては図で示すとおりである。

（3）restricted-ETSC から実行モジュールへの変換

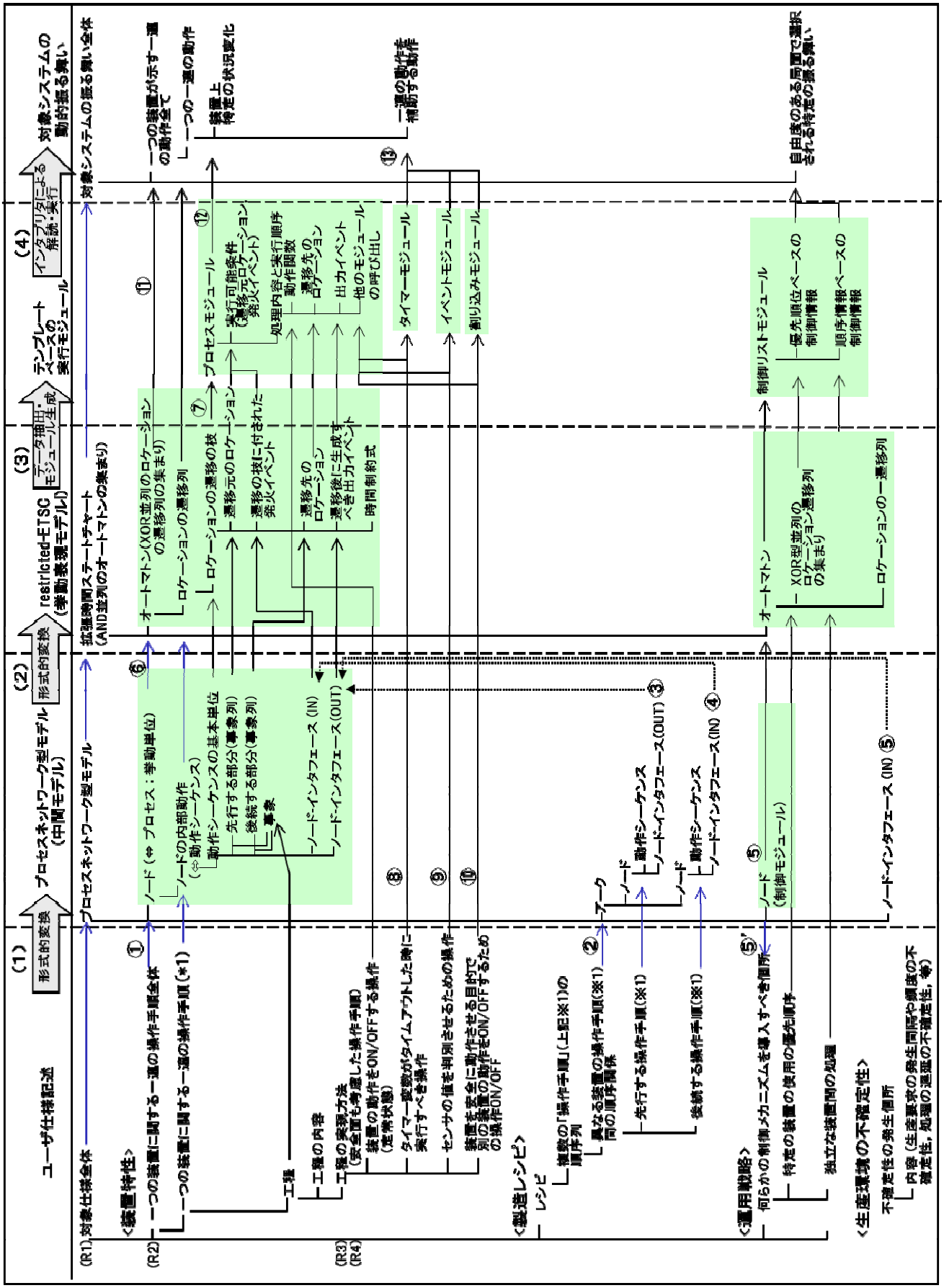
各ロケーションの遷移の枝はプロセスモジュールへ各々対応付けられる(図中⑦)。遷移の枝に付随する各要素からプロセスモジュールの内部データへの対応付けに関しては図で示すとおりである。また、プロセスネットワーク型モデルでは内部データとして位置づけられ明示的には表現されなかった「工程を実現する上で必要不可欠な細かい手順」が、プロセスモジュール以外の各種モジュールの内部データへ対応付けられる。具体的には、各工程を実現する手順の中にあるタイマー操作、センサ情報の監視、及び装置間の割り込みを伴う各動作の詳細が、各々、タイマーモジュール、イベントモジュール、及び、割り込みモジュールの内部データとして各々対応付けられる(図中⑧,⑨,⑩)。

（4）実行モジュールから対象システムの動的振る舞いへの対応関係

各オートマトンの記述が各装置上の一連の動作として実現され(図中⑪)、ロケーションの遷移の枝に対応付けられるプロセスモジュールが装置上の特定の状況変化として実現される(図中⑫)。また、その他のモジュールは、装置上の一連の動作を補助する動作として実現される(図中⑬)。

以上の説明から分かるように、Table4.1 に基づくと、本設計手法のベースモデルである拡張時間ステートチャートによる動作記述と制御システムの実際の動作とを、1 対 1 で対応付け可能である。そのため、実行中の実機の動作上の不具合箇所を、挙動モデル上で容易に同定することが可能である。このことから、Table4.1 は、設計段階だけではなく、制御系の変更時にも利用可能であると言える。以上のことから、本提案手法は、制御系のモデルベースでの設計が容易に行えるフレームワークを持つといえる。

Table 4.1:　モデル間の情報の流れ



以下では、上述のモデル同士の関係、及び、制御システムの振る舞いを、例を使って具体的に説明する。

4.5.2 制御系の動作メカニズムの詳細

Fig.4.24 は、各実行モジュールの設計過程を表し、Fig.4.31 は、得られた実行モジュールを入力データとして制御システムを駆動するためのコントローラの動作を表す。以下、(Ⅰ)実行モジュール設計のプロセス,(Ⅱ) 制御システムを駆動するためのコントローラの動作, の順に、具体的な手順の詳細を示す。

(Ⅰ) 実行モジュール設計のプロセス

先に述べたように、各実行モジュールを次のようにして作成する。即ち、ユーザ仕様記述(Fig.4.23-(D1))を埋め込みながらプロセスネットワーク型モデル(Fig.4.23-(D2))を作成し、それをETSC(Fig.4.23-(D3))に変換し、そのETSCからの情報抽出を各種テンプレートを用いて行うことにより実行モジュール(Fig.4.23-(D4))を生成する。以下、与えられるユーザ仕様記述、及び、各モデルへの変換の詳細について示す。

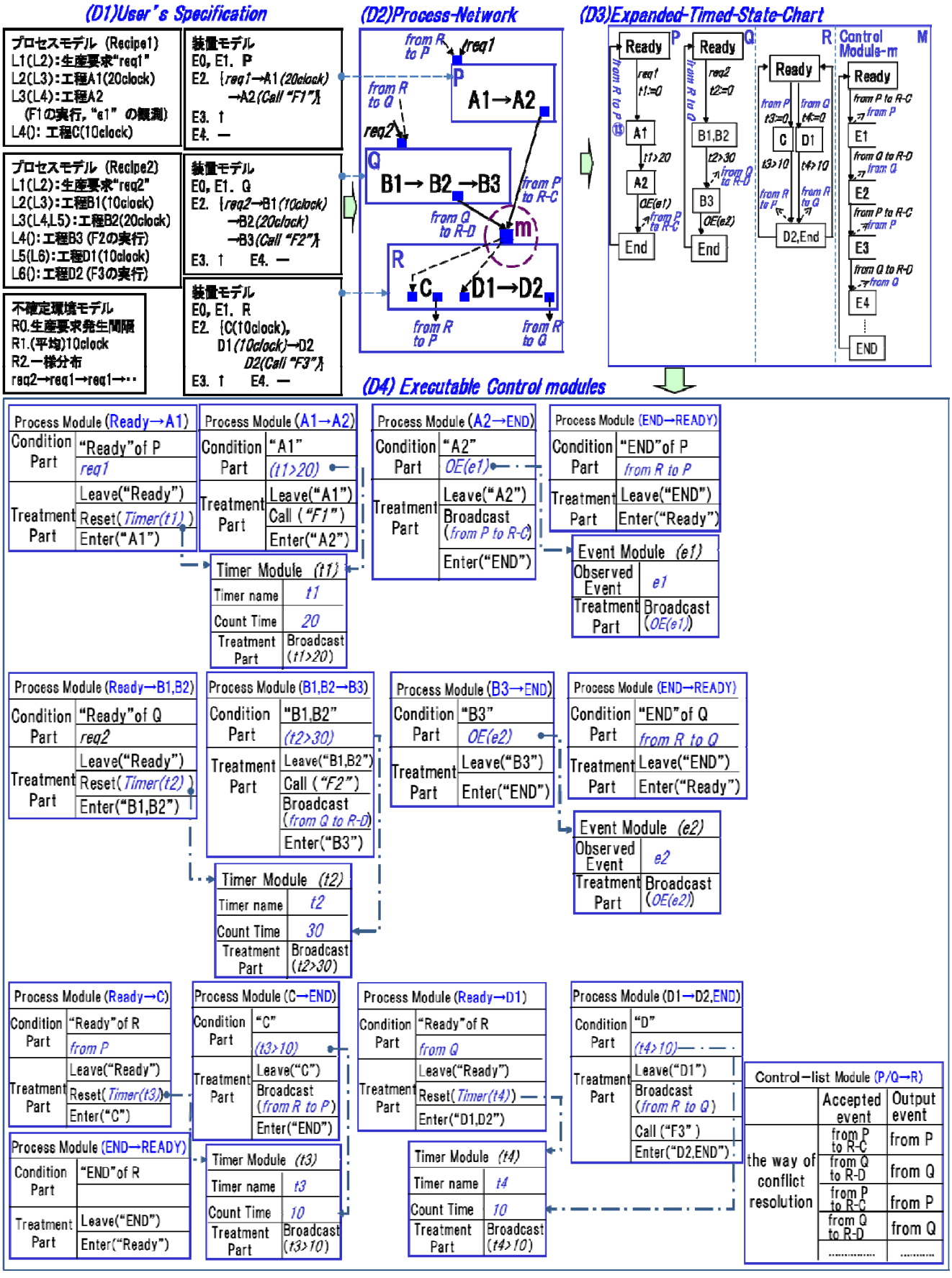


Fig.4.24: モデル同士の関係に基づくユーザ仕様から実行モジュールへの変換例

（1）ユーザ仕様記述（Fig.4.25 参照）

Recipe1，2 は製造レシピを表している。Recipe1 は、「生産要求“req1”が与えられる → 工程 A1(20clock) → 工程 A2（動作関数 F1 の実行，センサイベント“e1”がアクティブになる）→ 工程 C(10clock)」となっている。Recipe2 は、「生産要求“req2”が与えられる → 工程 B1(10clock) → 工程 B2(20clock) →（ 工程 B3（動作関数 F2 の実行） かつ（工程 D1(10clock) → D2(動作関数 F3 の実行))となっている。装置特性としては、装置 P で実現可能な振る舞い(動作シーケンス)は「req1→A1(20clock) →A2(Call “F1”)」、装置 Q で実現可能な振る舞いは「req2→B1(10clock) →B2(20clock) → B3(Call “F2”)」、装置 R で実現可能な振る舞いは「C，D1→D2」となっている。ただし、装置 R の E3 は 1、即ち、同時並行的に実行できるプロセスの数は高々1 である。従って、C と D1→D2 の間には、互いに排他的に実現されなくてはならないという制約が与えられていることとなる。また、製品の種類は 2 種類であり、生産要求の発生間隔は平均 10clock で正規分布し、生産要求は、req2→req1→req1→・・の順に与えられている。

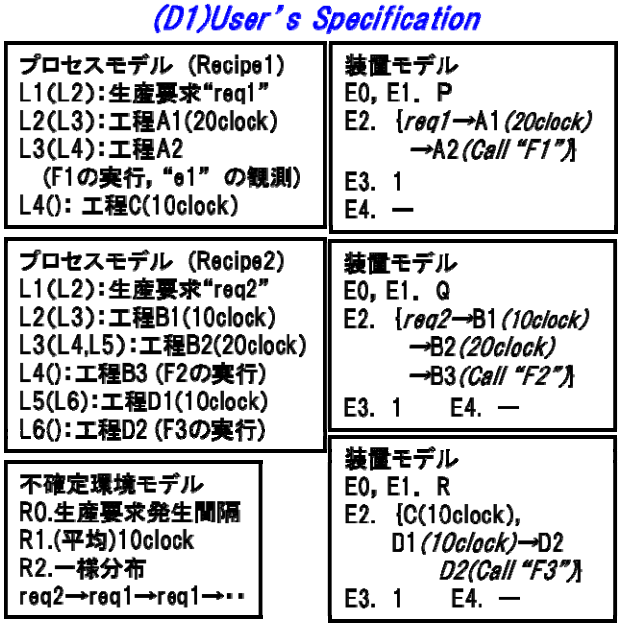


Fig.4.25: ユーザ仕様記述

（2）ユーザ仕様記述からプロセスネットワーク型モデルへの変換（Fig.4.26 参照）

ノードは、装置 P，Q，R の装置モデルと 1 対 1 対応付けて導入する。内部プロセスは、各装置モデルの E2（：動作シーケンス）に対応付けて各ノード内部に導入する。また、アークは、異なる装置上の工程間に与えられている順序に対応付けて導入する。本例では、プロセスモデル（Recipe1）の L3(L4)

が表す順序関係“L3→L4”に対し、PNM 上のアーク“A2 右端→m(R 上)”が導入されている。また、プロセスモデル（Recipe2）の L3(L4,L5) が表す順序関係“L3→L4”と“L3→L5”のうち、“L3→L5”に対しては、PNM 上のアーク “B2 右端→m(R 上)”が導入されている。なお、工程間の順序以外のデータ（処理時間や動作関数に関するデータ）は、動作シーケンスを成す各工程の内部データとして持たせ、PNM 上明示的には書かない。つまり、PNM では、処理の順序関係のみでシステムの外形を表現し、制御を要する部位をそのネットワークの構造的特徴から形式的手順により特定する。そして、処理時間や、要制御部位に埋め込む制御等、その他必要なシステム情報は、ETSC に変換した後に付加する。

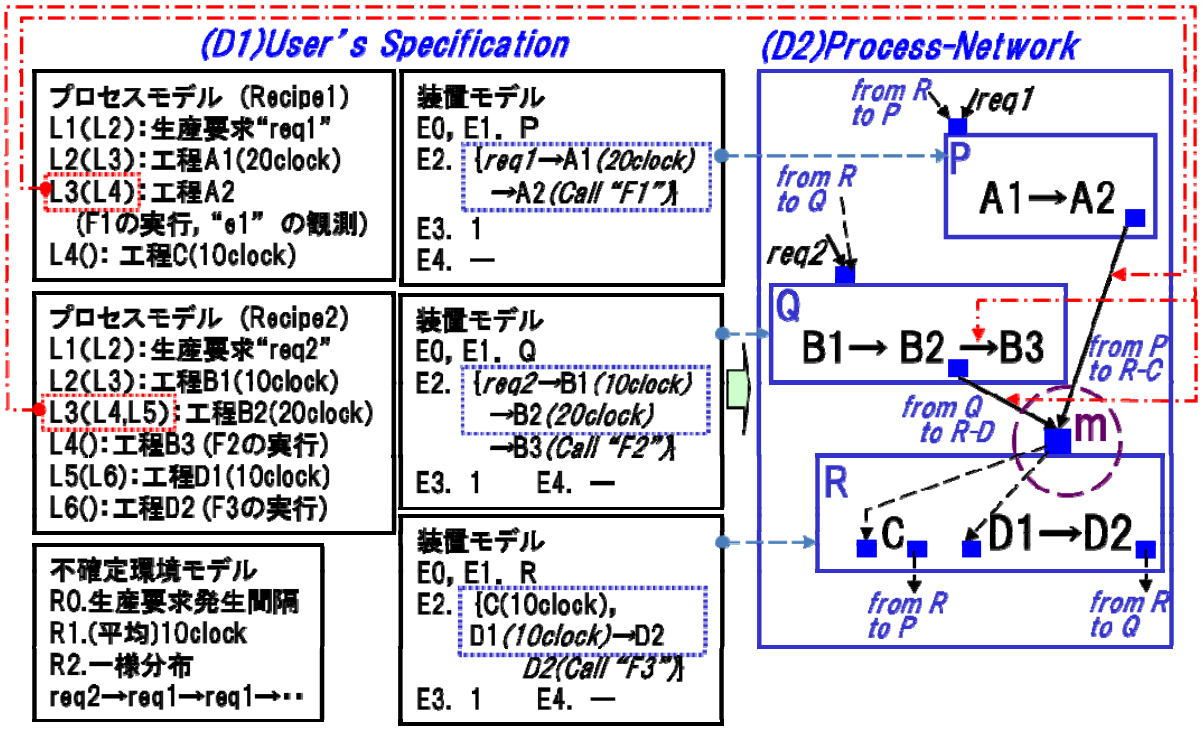


Fig.4.26: ユーザ仕様記述からプロセスネットワーク型モデルへの変換

（3）プロセスネットワーク型モデルから拡張時間ステートチャートへの変換（Fig.4.27 参照）

まず、PNM のノード P，Q，R に対応付け、順に、ETSC のロケーション P，Q，R を導入する。また、ノード P の内部プロセス｛A1→A2｝の動作シーケンス“A1→A2”に対応付け、ロケーション P 内部のオートマトンの 1 連のパス“Ready→A1→A2→End”を導入する。同様に、ノード Q に対応付け、ロケーション Q の内部のオートマトンが導入される。一方、ノード R の内部プロセスは互いに排他的に実行しなくてはならない 2 つの動作シーケンス C，D1→D2 を持つ。このような場合、各動作シ

ーケンスに対応付け、非決定的な 2 本のパスを持つ一つのオートマトンを生成する。本例の場合、ロケーション R 内部のオートマトンとして、非決定性を持つ 2 本のパス“Ready→C→D2,End”と“Ready→D1→D2,End”を導入する。なお、D2 と End が同一ロケーションにまとめられている理由は、D2 と End の間にイベントの発火も出力もないため、境界が必要ないためである。

次に、PNM のアークに対応付け、ETSC 上の生成イベントと発火イベントの対を導入する。本例では、PNM のアーク“A2 右端(in P)→m(on R) ”に対応付け、ETSC 上の P 上の生成イベント“from P to R-C”（遷移“A2→End”直後の生成イベント）と R 上の発火イベント“from P to R-C”（遷移“Ready→C”生起時の発火イベント）の対を導入する。アーク“B2 右端(in Q)→m(on R) ”に対しても同様である。

更に、PNM 上に明示的に書かれていない各工程の内部データ（処理時間、センサイベント等）を導入する。具体的には、工程の処理時間は、その時間を計るためのタイマー変数のリセット式と処理時間の満了をチェックするための制約条件式の組として表す。例えば、Recipe1 の工程 A1（on 装置 P）の処理時間(20clock)に対し、ETSC 上の P のロケーション A1 に入る遷移の矢印上のリセット式“t1:=0”と、A1 を出る遷移の矢印上の t1＞20 の組を導入する。また、同 P 上の発火イベント“OE(e1)”は、Recipe1 の「L3（L4）: 工程 A2 (F1 の実行，“e1” の観測)」の（センサイベント）「“e1”の観測」を表す。なお、動作関数 F1,F2,F3 は、実機を操作する際の命令である。そのため、ETSC モデルでは明示的に表示せず、内部データとして持たせておく。

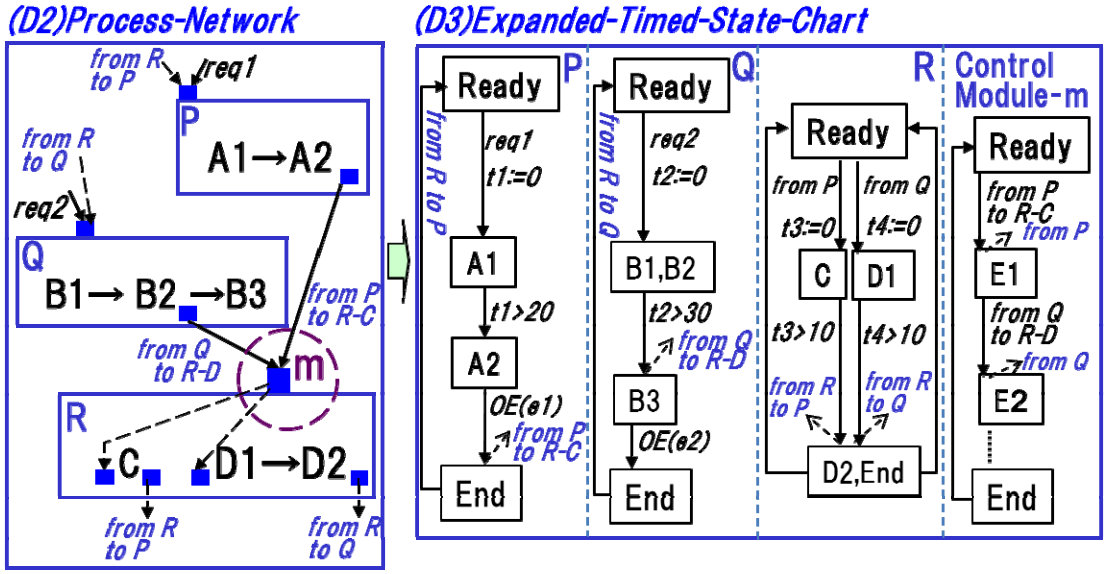


Fig.4.27:　プロセスネットワーク型モデルから拡張時間ステートチャートへの変換

(4) 拡張時間ステートチャートから制御系の実行モジュールへの変換 - プロセスモジュールの生成 -

(Fig.4.28 参照)

プロセステンプレートを使って ETSC から情報を抽出しプロセスモジュールを生成する。先に述べたように、プロセステンプレートに埋め込む情報としては、実行開始条件(アクティブであるべきロケーションとイベントの組)と処理内容(トークンの移動，イベントの生成，各種モジュール呼び出し)である。ETSC のロケーション P からの抽出は次にようにして行う。まず、ロケーション P は 4 つの状態遷移から構成されている。具体的には、Ready→A1， A1→A2， A2→End，End→Ready である。プロセステンプレートは、この 4 つの状態遷移各々に 1 対 1 対応付けて導入する。具体的に、Ready→A1 に関する情報抽出は次のようにして行う(Fig.4.28-①参照)。まず、プロセスの実行開始条件として、当該遷移が生起する上でアクティブであるべきロケーション“Ready”と、発火可能であるべきイベント”req1”を抽出し、プロセステンプレートの条件部(Condition Part)に埋め込む。更に、処理内容として、ETSC から抽出される情報は、ロケーション Ready から A1 へのトークンの移動、及び、タイマー変数ｔ1 のリセットである。具体的には、トークンの移動は Leave(“Ready”)と Enter(“A1”)の組で、ｔ1 のリセットは Reset(Timer(t1))という表現としてテンプレートの処理部(Treatment Pert)に埋め込まれる。

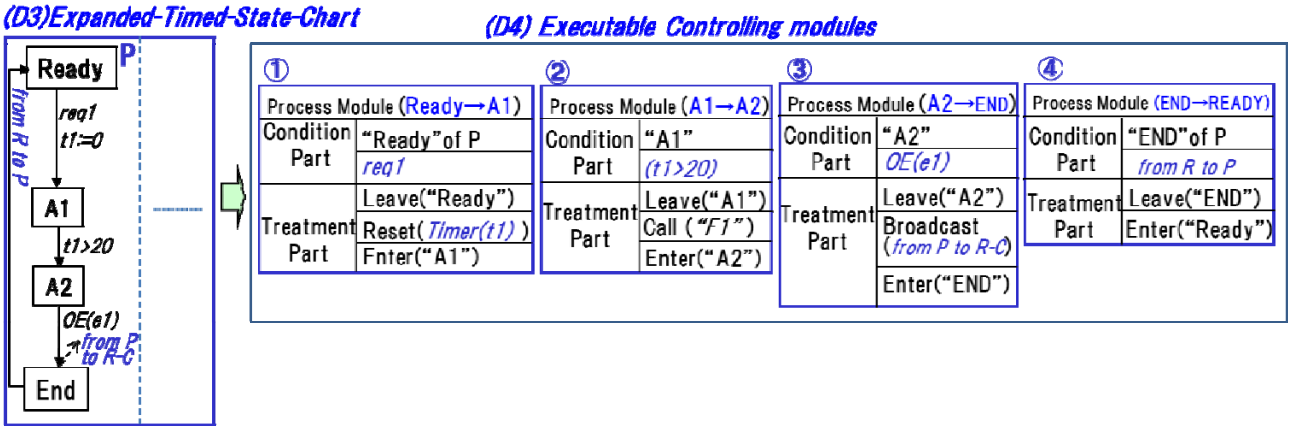


Fig.4.28: 拡張時間ステートチャートから実行モジュールへの変換-1 (プロセスモジュールの生成)

(5) 拡張時間ステートチャートから制御系の実行モジュールへの変換 - プロセスモジュールから呼び出されるその他の実行モジュールの生成 - (Fig.4.29 参照)

以上のように、プロセステンプレートは、ETSC のロケーションの遷移と 1 対 1 対応付けて形式的な手順に従って導入される。一方、タイマーモジュール、イベントモジュール、割り込みモジュールに関しては、自身以外の何らかのモジュールからの呼びだし命令(或いは、参照命令)と対応付けて導入されるが、いずれも形式的な手順に従って導入，生成される。

[例]（タイマーモジュールの導入，生成）

タイマー変数 t1 に関するタイマーモジュールは、次のような手順で、導入，生成される。まず、上記(3)で得られたプロセスモジュールの内部データにあるタイマー変数 t1 のリセット命令“Reset(Timer(t1))”(Fig.4.29 の①参照)を受け、タイマー変数 t1 のためのタイマーテンプレートを一つ導入し、“Timer name”の項に“t1”を与える(同図-⑤)。また、同図-②の“t＞20”を受け、当該タイマーテンプレートの“time out”の項に“20”を与える。更に、タイマー変数 t1 がタイムアウトした事実をそのタイミングで通知させるための命令“Broadcast(t1>20)”を処理部“Treatment Part”に与える。

[例]（イベントモジュールの導入，生成）

Fig.4.29 の⑥は、特定のセンサーが ON になることを条件に実行される“イベントモジュール”である。このモジュールは次のような手順で、導入，生成される。まず、プロセスモジュールの内部データにある、着目するセンサが ON であるときにアクティブとなる“OE(e1)””(Fig.4.29 の③参照)を受け、当該イベント e1 のためのタイマーテンプレートを一つ導入し、“Observed Event”の項に“e1”を与える(同図-⑥)。更に、イベント e1 がアクティブになった事実をそのタイミングで通知させるための命令“Broadcast(OE(e1))”を処理部“Treatment Part”に与える。

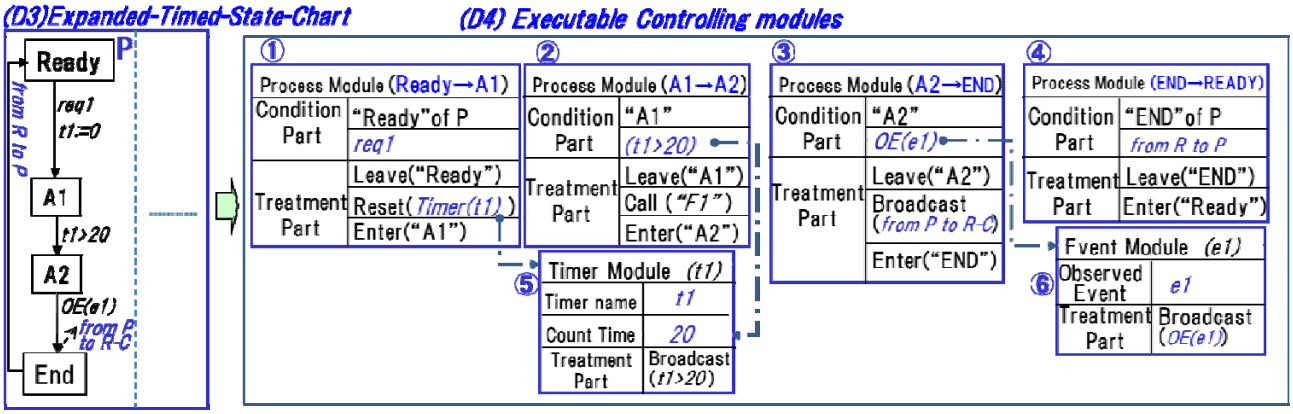


Fig.4.29: 拡張時間ステートチャートから実行モジュールへの変換-2
(プロセスモジュールから呼び出されるその他の実行モジュールの生成)

上述の例から分かるように、プロセスモジュール以外のモジュール（タイマーモジュール、イベントモジュール、割り込みモジュール）の生成では、テンプレートに埋め込むべき必要なデータは、ETSC の特定の部分にまとまって存在しているわけではなく、ETSC 全体に散在する必要な情報を検出し収集しなくてはならない。しかし、このような特定のデータの検出は、既に他のテンプレートで抽出された実行モジュール内部のデータを利用することにより、機械的な手順によって行える。具体的には、各データに付された特定の文字列や符号（Reset，不等号，OE，等）をキーとして行うことができる。Table 4.2 では、タイマーモジュール、イベントモジュール、割り込みモジュールを生成する上で利用可能な特定の文字列，符号について示す。

Table 4.2: 各種モジュールを生成する上で利用可能な特定の文字列や符号

Timer Module		
内部データの項目	データ抽出上キーとなる文字列，符号	書式
Timer Name	*Reset*	*Reset("Timer Name")*
	Suspend	*Suspend("Timer Name")*
	Resume	*Resume("Timer Name")*
Count Time	< ≦ =	*"Timer Name" < "a constant"*

Event Module		
内部データの項目	データ抽出上キーとなる文字列，符号	書式
Observed Event	*OE*	*OE("Event Name")*

Intervene Module		
内部データの項目	データ抽出上キーとなる文字列，符号	書式
Intervene Nama	Intervene	*Intervene("Equipment Name")*
	non-intervene	*non-intervene("Equipment Name")*

(6) 拡張時間ステートチャートから制御系の実行モジュールへの変換 - ETSC の制御モジュールから制御リストモジュールの生成 - (Fig.4.30 参照)

ETSC の制御のためのモジュール“Control Module-m”(以降 “M”) は、装置 P と Q の装置 R の使用の競合を解消するためのモジュールである。装置 R の競合解消のためのモジュール M に関する基本的な振る舞いは次の通りである。装置 P または装置 Q から装置 R の使用要求(“from P to R-C”(図中①)または“from Q to R-D”(図中②))が発生する度に、制御のためのモジュール M は、次に使用させるべき装置を発火イベントとの照合という形でチェックする。具体的には、１回目は、出された使用要求のイベントが、図中③と一致するかどうか、つまり、P からの使用要求であるかどうかをチェックする。２回目以降は、同様に、出された使用要求のイベントが、図中⑤, ⑦, ⑨と一致するかどうかをチェックする。そして、要求元の装置が次に使用させるべき装置と一致していれば、装置 R の使用を許可するイベントを生成しブロードキャストする。例えば、装置 R の１回目の使用要求として、装置 P からの使用要求イベント(図中①)が生成されると、それがモジュール M 上のイベント③と一致し、当該イベントが発火し、装置 P への使用許可イベント(図中④)が生成され、装置 R の振る舞いを表すロケーション R 上で、装置 P からの要求を受けた処理（ロケーション“C”で表される工程の処理）が行われ、その処理時間“10”が満了すると、装置 R 使用終了を表すイベント(図中⑫)が出され、装置 P 側の処理も終了する。一方、１回目の使用要求として、装置 Q からの使用要求イベント(図中②)が生成されると、それがモジュール M 上のイベント③と一致せず、当該イベントは発火しないため、装置 Q への使用許可イベントは生成されない。以上のような振る舞いを繰り返し行うことが、ETSC 上でのモジュール M の競合解消の仕方である。

ETSC の制御のためのモジュールが示す順序情報を抽出し、制御リストテンプレートに埋め込むことにより制御リストモジュールは生成される。抽出すべき順序情報とは、発火イベントと生成イベントの組の順序列として抽出できる。本例では、(使用要求の受理, 使用許可の生成と出力)の組の順序列であり、具体的には、 (③, ④), (⑤, ⑥), (⑦, ⑧), (⑨, ⑩)の順序列である。そして、これら順序列のデータを制御リストテンプレートに、与えられたフォーマットに従って埋め込む。即ち、(受理イベント, 出力イベント) の組の順序列として与える。その結果得られた制御リストモジュールは Fig.4.30-(D4) の通りである。

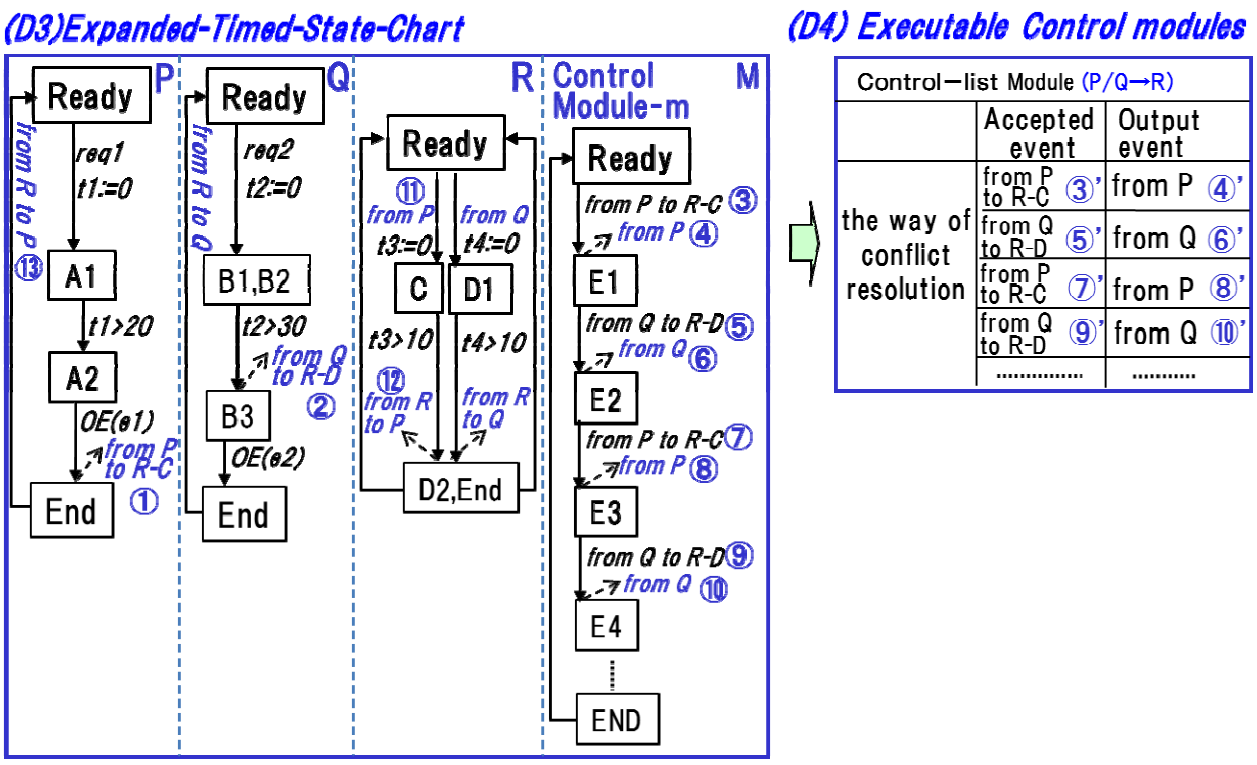


Fig.4.30:　拡張時間ステートチャートから実行モジュールへの変換-3
(制御リストモジュールの生成)

（Ⅱ）制御システムを駆動するためのコントローラの動作

（Ⅰ）のように得られた実行モジュール群を制御システムへ与え、コントローラにより実行させる。先に述べたように、制御システム内部の実行の流れは次の通りである(Fig.4.31 参照)。まず、各プロセスモジュールの条件部の真偽を現在のロケーションやイベントのＯＮ／ＯＦＦに基づきチェックする。そして、条件部の条件が全て成立しているプロセスモジュールの処理部、及び、そのようなプロセスモジュールから呼び出される実行モジュールの処理部を処理するためのスレッドを個々のモジュールごとに作成する。これらはマルチスレッド方式で並列処理させる。なお、ここで並列処理される個々の処理とは、具体的には、挙動モデル上の各ロケーションやイベントと１対１に対応付ける形で導入したロケーションやイベントのＯＮ／ＯＦＦの更新、実機の各パーツの動作(ドライバープログラム)に対応付けられる動作関数のＯＮ／ＯＦＦの更新、及び実行である。以下、各振る舞いの詳細を示す。

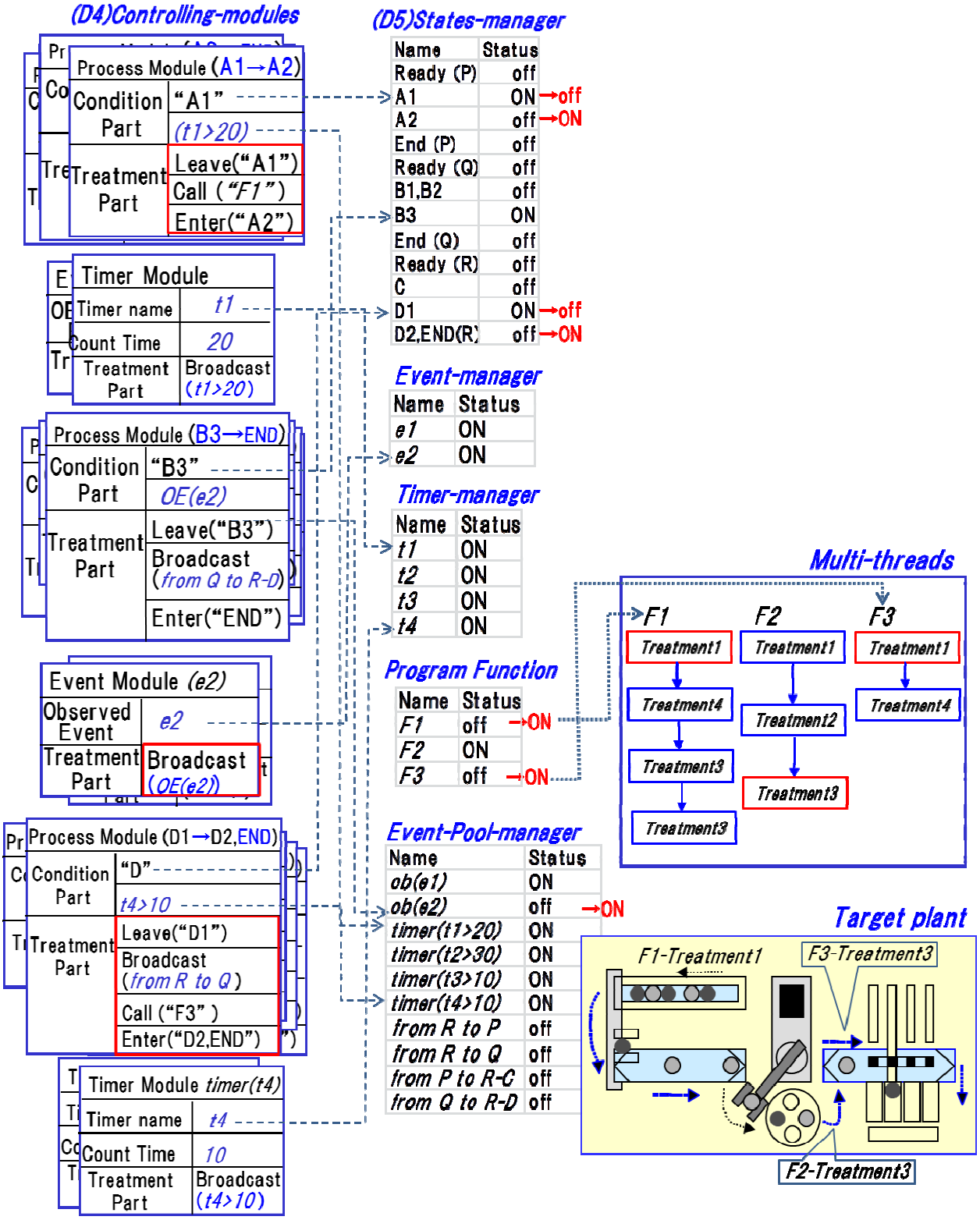


Fig.4.31: 制御システムを駆動するためのコントローラの動作

（1）プロセスモジュールの条件部の真偽のチェック（Fig.4.32 参照）

プロセスモジュール“B1,B2→B3”を例に説明する。Fig.4.32 は、本説明に関わるプロセスモジュールと、システム内部状態を管理するための各種表である。表に書き込まれた値全体の組が、ある時点でのシステム内部状態を表す。本モジュールがアクティブであるための条件は、ロケーション“B1,B2”がアクティブであり(図中①, ②)、時間制約“t2＞30”が満たされていること(図中③)である。本例では、図中、各ロケーションの真偽を管理する State-maneger において、ロケーション“B1,B2”は ON(アクティブ) (図中④)である。また、各タイマー変数の値とタイムアウト状況を管理する Timer-manager において、t2 は ON（タイムアウトしている,）である。つまり、時間制約“t2＞30”が満たされている。以上のことから、プロセスモジュール“B1,B2→B3”の条件部は真と判断され、この後、処理部が実行される。

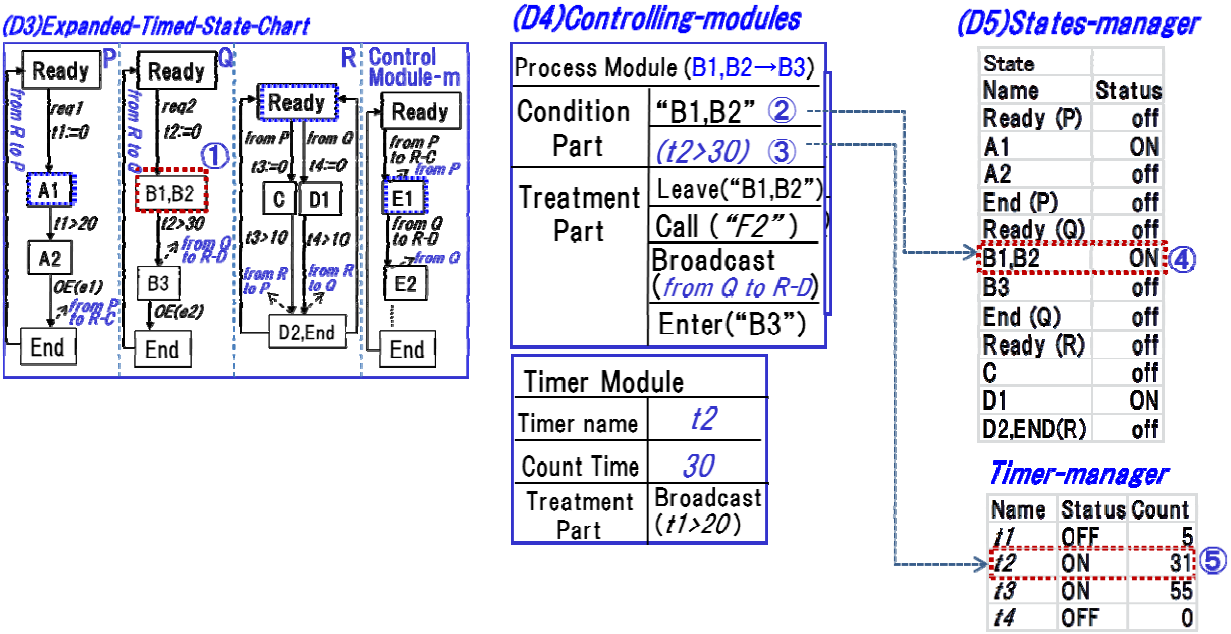


Fig.4.32:　プロセスモジュールの条件部の真偽のチェック

（2）プロセスモジュールの処理部の実行（Fig.4.33 参照）

プロセスモジュール“B1,B2→B3”の処理部の実行について説明する。まず、ロケーション“B1,B2”を OFF とする(図中①, ②)。次に、動作関数 F2 をアクティブとする(図中③)。更に、装置 R の使用要求イベント“from Q to R-D”をアクティブとする(図中④)。最後に、ロケーション“B3”をアクティブ(ON)にする(図中⑤, ⑥)。

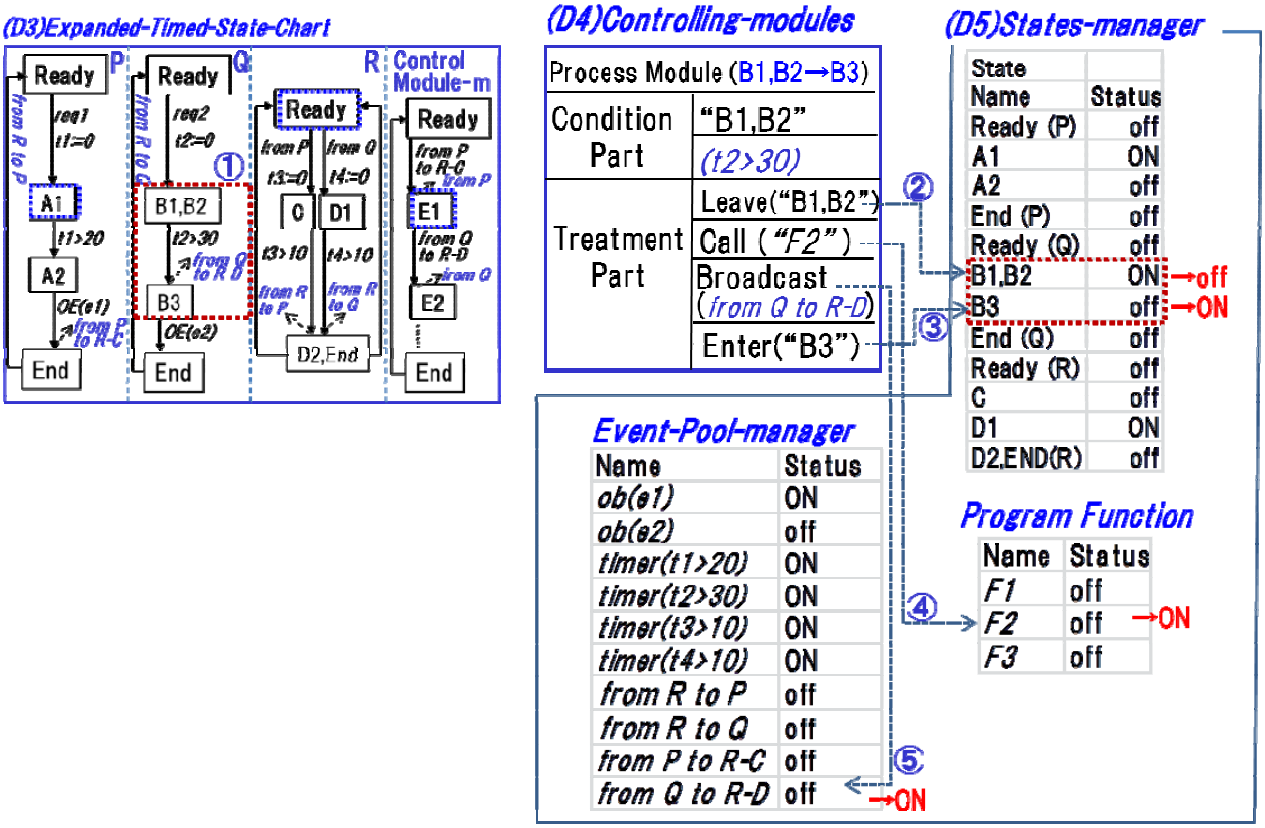


Fig.4.33:　プロセスモジュールの処理部の実行

（3）実行モジュールの並列処理

複数のモジュールの条件部が真である場合、処理部の実行手順を表すスレッドを個々のモジュールごとに作成し、これらをマルチスレッド方式で並列処理させる。

[例]　2 つのプロセスモジュールの並列実行　(Fig.4.34 参照)

Fig.4.34 は、プロセスモジュール“A1→A2”の条件部(図中①，③)と“B1,B2→B3”(図中②，⑤)の条件部が真となり、各々の処理部に対するスレッドを作成し並列実行するケースを示している。具体的には、図中の処理部④に対してスレッド⑦を、処理部⑥に対してスレッド⑧を生成し、並列実行する。

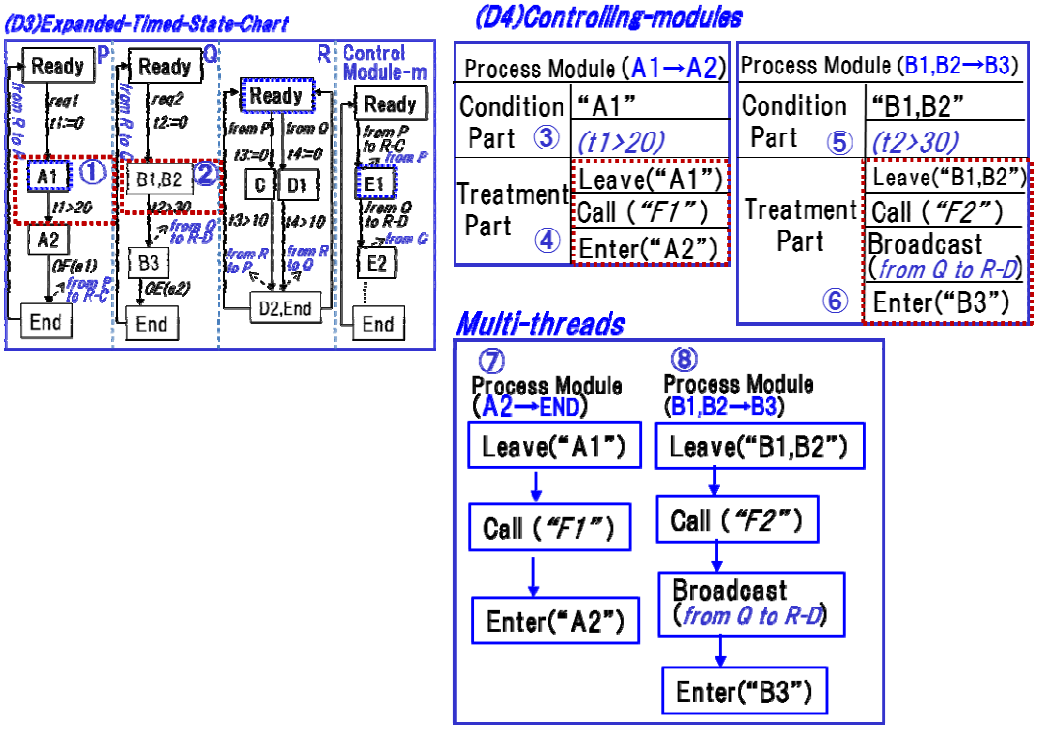


Fig.4.34:　2 つのプロセスモジュールの並列実行

[例]　新たなモジュールの実行開始　(Fig.4.35)

実行開始条件の整ったモジュールが新たに発生すれば、その処理部を処理するためのスレッドを新たに追加する。本例では、イベント“e2”がアクティブになったことを受け、イベントモジュール“e2”(図中①)の実行が開始し、その処理部を処理するためのスレッドが追加されている(図中②)。なお、この時、実行中の他のスレッドはそのままである。

[例]　モジュールの実行終了　(Fig.4.35)

処理の終わったスレッドは削除する(図中③)。この時、実行中の他のスレッドはそのままである。

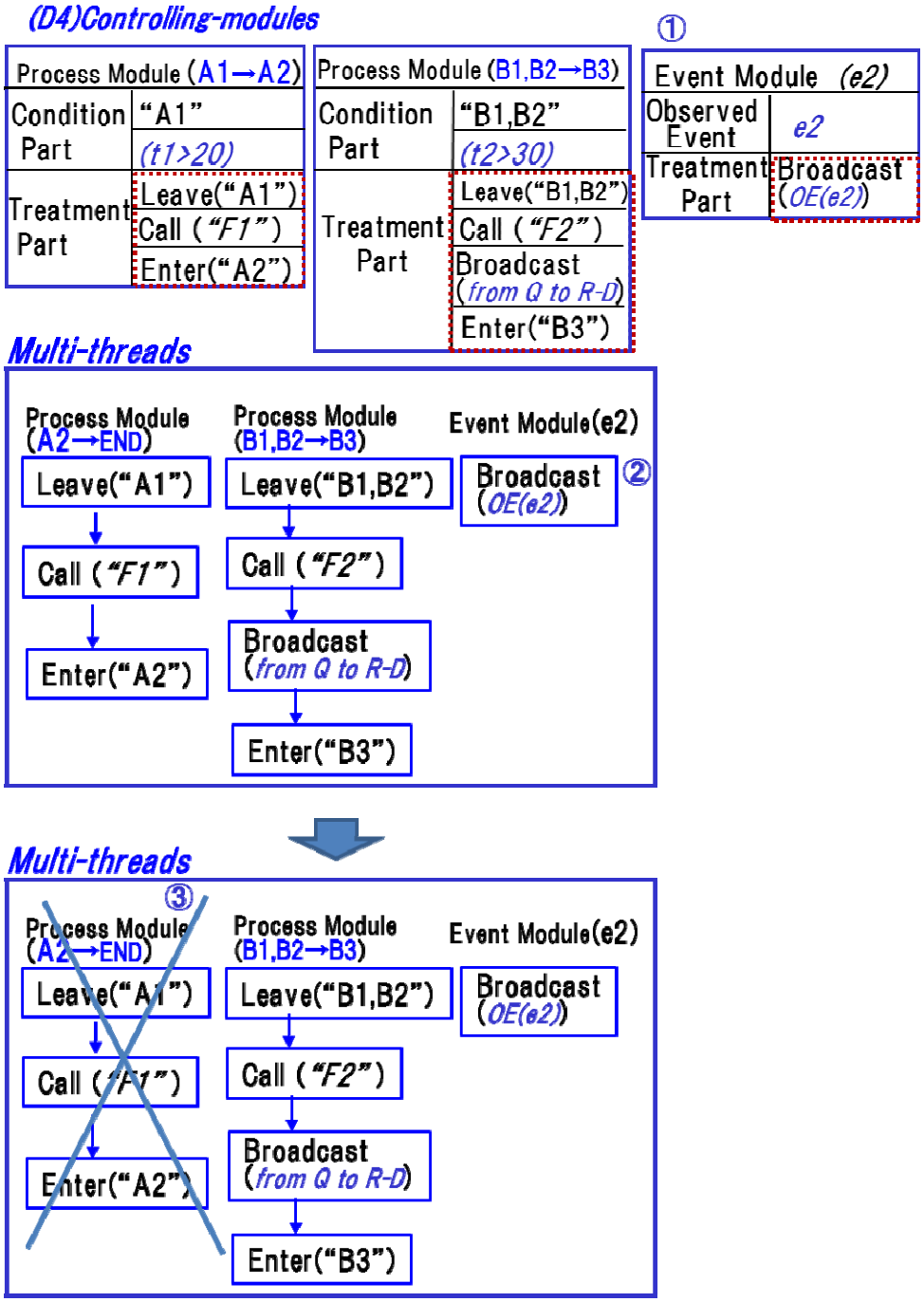


Fig.4.35:　2 つのプロセスモジュールの並列実行

4.5.3　大きな不確定性にも動的対応可能な制御系の実現方法

再スケジューリング結果から与えられる制御要求への動的な切替えでの処理の流れは以下の通りである。

(q1)スケジュール変更事由の発生タイミングやその内容に基づき、計画変更すべき領域を、スケジュール(ガントチャート)上で特定する。

(q2) 計画変更領域に対する再スケジューリングをし、変更領域に対する新たな制御リストを作成する。

(q3)元の 制御リストから変更すべき部分を一旦切り出してキャンセルし、代わりに(q2)で得られた新たな制御リストの内容を追加する。

以下、提案事項である(q3)の実現方法について示す。

＜制御リストの管理方式＞

コントローラでは、制御リストは、CLM(：Control-List Manager) という管理主体により次のように管理する。

・準備段階では、コントローラに与えられた制御リストモジュールは全て、CLM により、CLP(：Control-List Pool) という制御リストの格納領域に各々独立に格納される。また、各制御リストの要素は (Input, Output, Flag) の 3 つ組データから成り、要素の並び順は実行順序を表す。ここで、要素 (Input, Output, Flag) を読み下すと、「(次の装置操作を要請する)イベント Input が発火したならば、(その要請に対するリアクションとしての)イベント Output を出力せよ」となる。ここで、Output の内容は、特定の装置に関する使用許可を、当該装置を次に使用させる主体に対して与えるためのメッセージとなっている。このように、各装置の使用順序は、各々対応付けられる制御リストの Output の並び順として記述される。また、Flag の値は当該要素の実行前は 1 であり、実行後は実行済みを意味する 0 へ、CLM によって書き換えられ。

・実行段階では、ある実行モジュールが、特定の制御リストに対する参照要求を出すと、CLM がその要求を受け取り、CLP に登録された制御リスト中から指定のリストを特定する。そして、当該制御リストの未だ実行されていない要素(即ち、Flag の値が 1 の要素)のうちの先頭の要素を実行し(参照させ)、その要素の Flag は 0 に書換えられる。

＜制御リストの動的変更の仕方＞

まず、計画変更すべき制御リスト(以降“変更対象リスト”) の変更対象領域に関する新たな制御リスト(以降“更新用リスト”)を用意する。次に、実機を止めないまま、コントローラに上記の更新用リストを読み込ませる。(読み込ませた制御リストは直ちに CLP に登録されるため、この時点では CLP に

は変更対象リストと更新用リストが重複して存在している。)　その直後に、変更対象リストに対し、以下の処理を行う。

(1)まず、変更対象リストの変更すべき要素（先頭から i 番目～j 番目(i≦j)）の実行をキャンセルするため、付された全てのフラグの値を強制的に 0 にする。

(2)次に、全ての要素の Flag の値を 1 とした更新用リストを、その先頭が j+1 行目となるように変更対象リストに追加する。

[例 4.11] (制御リストの変更手順)

Fig.4.36-(1),(2)は、順に、ある生産プラントに対する初期スケジューリング結果(Fig.4.36-(1))、及び、その結果から得られる制御リスト(Fig.4.36 -(2))を表す。そして、Fig.4.36 -(3)～(7)は、計画変更事由の発生を受けた一連の処理の流れを説明する図になっている。具体的には、以下の通りである。

Fig.4.36 -(3)：　再スケジューリング領域の切り出しと制御リスト上の変更領域の特定（変更領域の要素の Flag の値は全て 1 となっている。）

Fig.4.36 -(4)：　制御リストの変更領域の Flag の値の変更(1 から 0 への変更)による実行のキャンセル

Fig.4.36 -(5),(6)：　再スケジューリング結果の制御リストへの追加（Flag の値は全て 1 に設定しておく。）

Fig.4.36 -(7)：　制御リストの変更完了

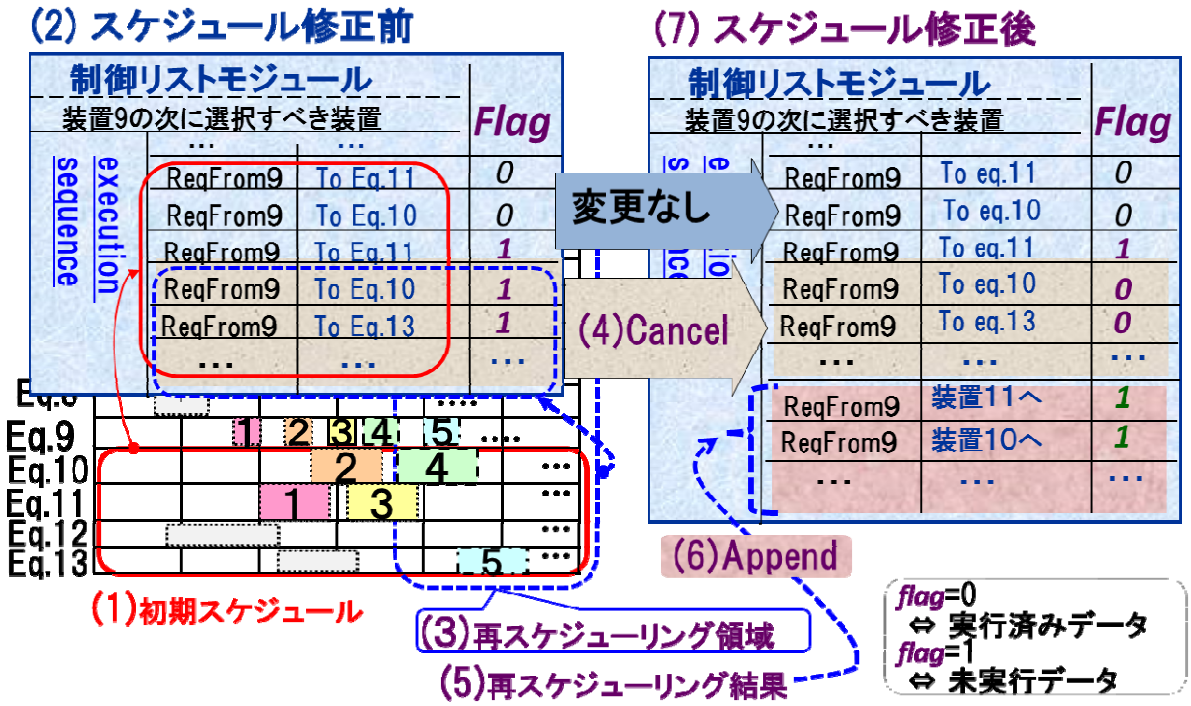


Fig.4.36：　制御システム実行中の制御リストモジュールの部分的な変更の仕方

4.6 検証実験 （- 制御系設計手法の妥当性の検証）

提案する方法論の妥当性を検証するために、フローショップ型の FA 実験装置(Fig.4.37(上図))を制御対象として実験を行った。実験の目的は、以下の 2 点を確認することである。

(1) 提案する制御系設計のためのフレームワークの妥当性

確認の仕方としては、先ず、本稿で提案してきた、各ステップでのデータ変換方法に基づき、制御系設計に必要なデータをフレームワークの上流から下流へ順々に受け渡し、実行モジュールを生成する。そして、得られた実行モジュールを実機のコントローラへ入力し、実機を動作させ、それが最上流で与えるユーザの要求仕様通りの動作であるかどうかを確認すること。なお、実行モジュール作成までのデータの変換は、具体的には、要求仕様から、中間モデル「プロセスネットワーク」を経由させ、実行制御対象の要求動作を表す ETSC (Fig.4.37(下図))を作成し、実装したモジュール抽出システムへ入力する、という手順で行う。

(2) 処理の遅れへの対処方法の妥当性

①外乱が小さいとき、少しずれはしても、スケジューリング結果が示す最適値に近い動作が実現できること。(時間駆動のスケジュール情報を事象駆動情報の制約条件に変換して使用すれば、小さな不確定変動が一定の効率低下を許容することで吸収できるということ。)

②外乱が大きいとき、再スケジューリング結果への動的な切り替えが実現できること。(再スケジューリング結果の実行系への受け渡し方法が妥当であること。)

具体的に、②では、理想的な動作である当初のスケジューリング結果に対し、大きな不確定性の発生を想定し、再スケジューリング結果への動作の変更が、提案方法に従って実機を止めることなく動的に行えるかどうかを確認した。本実験装置は、Fig.4.37 で示すような 2 工程のフローショップ型生産システムである。

具体的には、ターンテーブルを第 1 工程、ストッカーを第 2 工程と見立てて、それぞれに処理能力の異なる 4 つの加工装置(第 1 工程：T1,T2,T3,T4、第 2 工程：S1,S2,S3,S4) があるものと想定して各ジョブの処理時間を与えて実験を行った。処理対象であるワークの一連の処理の流れは、パーツフィーダからピック＆プレース、搬入コンベア、ターンテーブル、搬出コンベア、ストッカーの順となる。ただし、本装置では、「搬入コンベアからターンテーブルへ」と、「ターンテーブルから搬出コンベアへ」の双方のワーク移送で同一のアームロボットを使用しており、アームロボットで発生する競合解消のための制御が必要である。

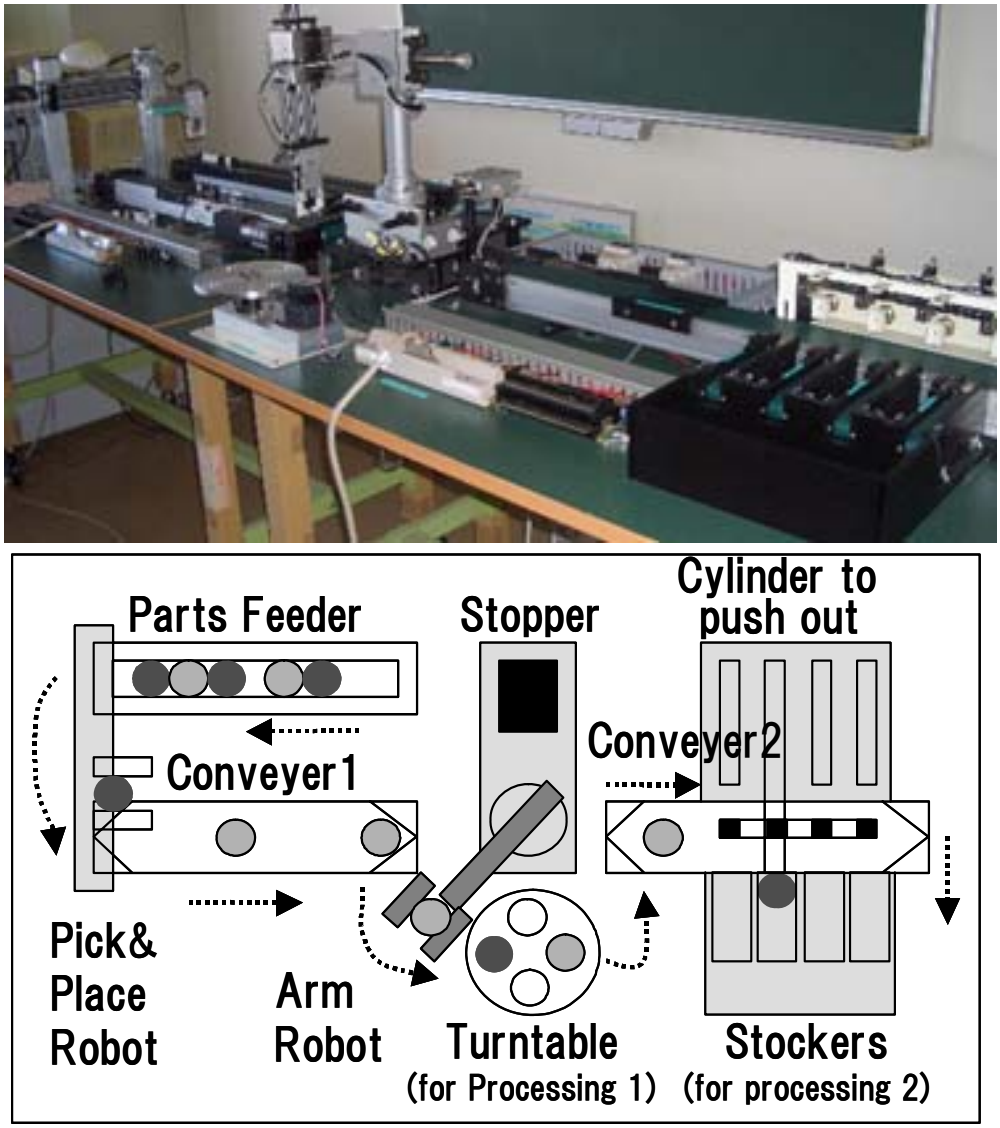


Fig.4.37:　ミニ FA プラントの概観と挙動表現モデル ETSC の階層性最上位

4.6.1　実験１　（- 制御系設計のためのフレームワークの妥当性検証）

まず、FA プラントの振る舞いを表す ETSC モデルをプロセスネットワーク型モデルを経由して生成し（Appendix-A1.1，A1.2，A1.3 ）、その ETSC をモジュール抽出システムへ入力し、実行モジュールへの変換を行った。その結果、プロセスモジュール 142 個、タイマーモジュール 9 個、割り込みモジュール 2 個、イベントモジュール 1 個、及び、制御リストモジュール 4 個から成る都合 158 個の動作モジュールが生成された(Appendix-A1.4)。そして、求める制御を実現するための数値パラメータが埋め込まれた、以上のような 158 個の実行モジュールを入力データとし、FA システムを起動した。そして、FA システムの実際の動作は、ETSC モデルによる要求仕様動作に従っているということが、モデルをトレースすることにより確認できた。以上のことから、提案するデータ抽出手法は正しく機能している

ことが確認できた。(Fig.4.38)

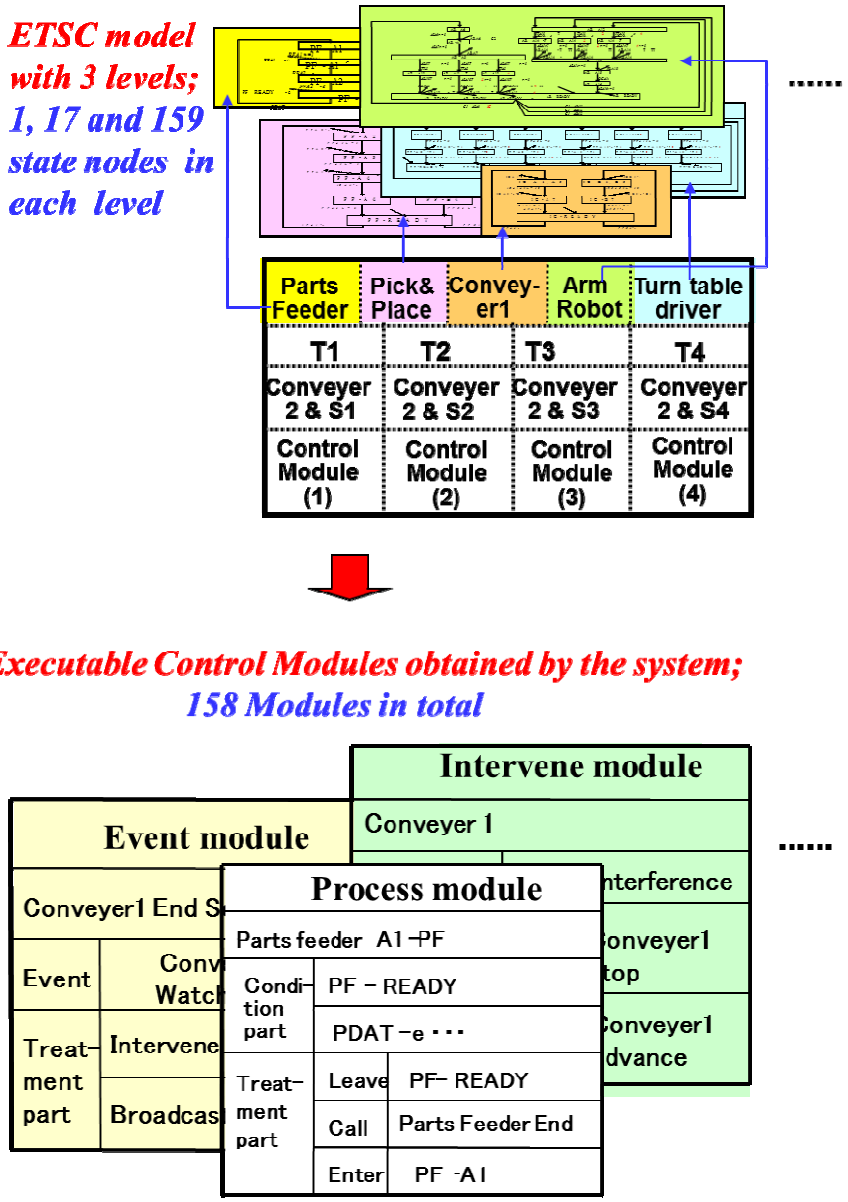


Fig.4.38： ミニ FA プラントの挙動表現モデル ETSC の階層性最上位（上図）
とモジュール抽出システムで得られた制御系動作モジュール(下図)

4.6.2 実験２ （- 処理の遅れへの対処方法の妥当性）

（I） 処理遅延発生のケース

提案するフレームワークの下で今回構想・実現した、実行時間の遅延(もしくはずれ)への対処方法が有効かつ妥当なものであることを、実機動作の実測結果とスケジューリング結果との一致性を示すことにより、確認した。その確認方法としては、変更後のスケジュールで示される(スケジュールから読み取れる)装置割り当てと装置使用順序が、実機の上でも(変更後のスケジュールと)同じ内容で実現でき

ているかどうかを確認するという方法を採った。従って、主たる実験データのひとつとしては、入力されたスケジュールと、出力である実機の動作を表した、2 つのガントチャート（Fig.4.39　スケジュールと実機の実測結果のガントチャート表示)があげられる。なお、実機動作に関しては、装置割り当てと装置使用順序については実機の動作を目測で観測・確認し、各処理の各所要時間については、実際にタイマーを使って計測した実測値を用いてガントチャートを作成した。以上のような検証実験を、スケジュール変更箇所を変えながら幾つかのケースで実施した。

一例として、「実行中、ターンテーブル上の加工装置の一つである装置 T4 で、現行の制御が継続不可能な位大きな遅延の発生が検知された。」という不確定性の発生を想定して行った実験について、Fig.4.39 と Fig.4.39 を使って説明する。ここで、Fig.4.39 の(S1)と(S2)，及び、(S3)と(S4))は、順に、本実験での初期スケジュールとそれに従って動作した実機の振る舞いの実測値、及び、変更後のスケジュールと同じく実機の実測値を表すガントチャートである。また、Fig.4.40 の＜A＞と＜B＞は、順に、スケジュール変更前の制御リストと変更後の制御リストの実データである。ただし、本図では、本対象が持つ 4 つの「制御を要する箇所」のうちの一つである「アームロボットに関する制御リスト」（“List1”）を示している。

まず、当初のスケジューリング結果(Fig.4.39-(S1))に従って制御リスト(Fig.4.40-＜A＞)を作成すると共に、本制御系設計手法に従って制御系の実行モジュール全体を生成し、それらをコントローラへ入力し実行した。そして、実行中のある時点で装置 T4 の大きな遅延が検知され、再スケジューリングが必要と判断されたとし、再スケジューリングの対象領域の始点(Fig.4.39-(S3)-P)を決定し、再スケジューリングを実施した。そして、その結果得られた再スケジュール（(Fig.4.39-(S3))）から装置割り当てと割り当て順序を取り出し、事象駆動の制御(制御リスト)を作成し、変更前の制御リスト(Fig.4.40-＜A＞，或いは＜B＞の i6～i3 行）に変更部分の新リスト(Fig.4.40-＜B＞- r1～r8 行)をマージし、新たな制御リスト(Fig.4.40-＜B＞)を作成した。更に、新制御リストの変更領域の旧データ（＜B＞の List1 の i6～i3 行）の Flag の値を強制的に 0（実行済み)とし、同リストの新データ(＜B＞の List1 の r1～r8 行)の Flag の値を 1(未実行)とした。以上のような制御の動的変更実験を、実機を止めないで行った。その結果、得られた実機の振る舞いを表すガントチャートが(Fig.4.39-(S4))である。実際、実機動作 Fig.4.39-(S4)は、制御リスト Fig.4.40-＜B＞が示す操作順序に従った振る舞いになっていることが分かる。なお、スケジュールと実測値(Hig7-(S1)と(S2)、或いは(S3)と(S4))の間には 0.3～0.4%程度の誤差があるが、これは、処理対象(ワーク)を装置(コンベアや加工装置)に乗せる、或いは、取り除くための所要時間を、スケジューリング段階では（通常は）考慮しないために生じる誤差である。

以上の具体例で示すリスト操作から分かるように、本稿で提案する「動的なスケジュール変更」の方法は、スケジューリング変更のタイミングが何時であっても、変更後の制御リストの追加とフラグの値の機械的な書き換えのみで動的な移行が行える方法である。実際、本方法を用いたスケジュール変更の検証実験を、変更箇所と内容の異なる複数のケースで試みた結果、スケジュール変更箇所や変更内容に関わらず、機械的に、再スケジューリング結果の受け渡しが可能であることが確認された。

また、外乱が小さいケースについても、他の条件設定等は上記実験と同様にして、スケジュール(制御リスト)の異なる幾つかのケースについて実験し、もとの最適スケジュールからは、少しずれはするものの、その最適値に近い動作を実現できることを、上記実験と同様の方法により確認した。

（Ⅱ）装置の故障発生のケース

次に、特殊なケースとして、特定の装置が故障した場合のスケジュールの変更も、制御リストの変更だけで、(Ⅰ)のケースと同じ様に行えるかどうかの確認実験を行った。具体的に、今回は、「急きょ装置 S2 が使用できなくなった」という不確定性の発生を想定した実験を行った。ここでは、装置 S2 と同じ機能を持つ他の装置を代用することを許して再スケジューリングを行った。他の条件設定等は、実験 1 と同様の流れで実験を行った。その結果、初期スケジュールでは S2 に割りつけられていた処理の全てを S4 に割りつけるという再スケジューリング結果が反映された制御リストが得られた。更に、その後の制御リスト変更処理は、(Ⅰ)の遅延のケースの場合と同様に、新リストを旧リストにマージする機械的な操作だけで、再スケジューリング結果が正しく実現されることを、先の実験と同様の手順により確認できた。

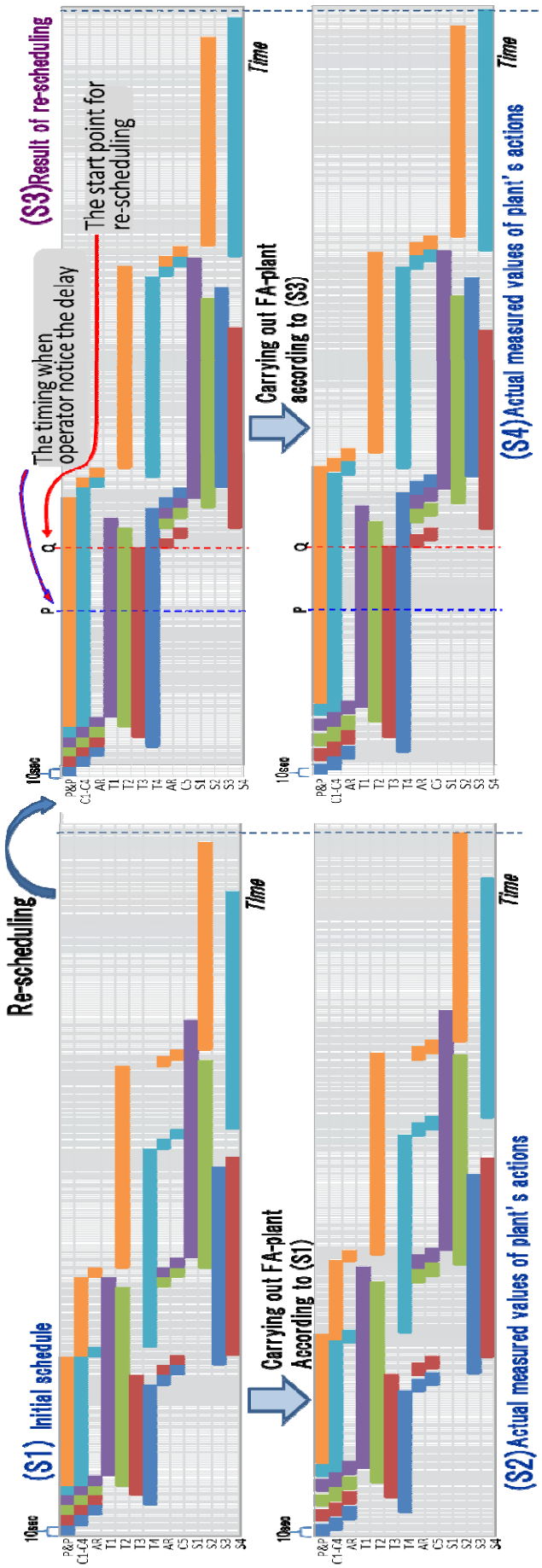


Fig.4.39:　スケジュール((S1),(S2))とスケジュールに従ったプラント動作の実測値((S3),(S4))との比較

Fig.4.40： 制御リストモジュール（初期リスト（A）、変更後のリスト（B））

4.6.3　実験の考察

4.4 節で提案したデータ抽出手順を用いることによって ETSC モデルによる要求仕様動作から実機を動かすための実行モジュールを自動的に得られることが確認できた。また、再スケジューリング結果に従った振る舞いを、制御システム自体を修正・変更することなく、入力データである制御リストを変更するだけで、かつ、その変更には関わりなく当初の制御に基づいて動作すべき他の部分の制御動作を妨げることなく、実現できることを確認した。

以上のことから、本手法が適用可能であるような特性を持つプラントに対しては、本フレームワークに沿って制御系の開発を行うことにより、従来の人手を介した、属人的技能・知識に大きく依存した方法が抱える非効率性を軽減できる可能性があることを、実証的に確認できた。

4.7　4 章のまとめ

不確定性を有する離散型生産システムの計画系での最適スケジューリング結果を、実行系へ系統的に受け渡すためのフレームワークを提案した。また、本フレームワークに基づくと、再スケジューリング結果に基づく変更を、対象プラントを止めずに動的に実行することが、簡単な技術的工夫により実現できることを示した。更に、上記の動的変更を自動的に行わせるためのプログラムを実装し、FA システムを適用例とした検証実験を行い、本アプローチの妥当性と有効性を確認した。

第 5 章　拡張時間ステートチャートに基づく検証手法

処理時間の不確定性や操作上の非決定性によって生じる(生産計画からの)ずれにシステムが何処まで耐えられるかといった問題に対し、これまでテストデータを使ったシミュレーション程度のチェックしか行われてこなかったという現状がある。このことを踏まえ、第 5 章では、離散型並列生産システムの適応限界を調べるための検証方法を提案する。具体的には、離散型並列生産システムの挙動表現モデルのベースモデルとして採用した TSC で使用されるモデルチェッキング型の動作検証法を、拡張時間ステートチャート(ETSC)に適用できるようにするために拡張する[38)~40),43)]。

拡張内容は次の通りである。既存のこの種の検証手法、即ち、Kripke 構造全体を時間領域分割して状態ノードを作成し、その上での検証を行う検証手法[1),3)]は、元々計算量爆発の問題を持っている。そのため、現実規模の生産プラントにそれを用いることは困難である。そこで、本研究では、対象プラントの可能な全ての振る舞い、即ち、“可能世界” に対し、(1)不確定性存在箇所の限定、(2)不確定性の有限個の非決定性への置き換えによる集約近似(注 5.1)、(3)イベントプール上での即時発火ルールの採用、という 3 つの制約を与える。そして、それによって限定生成された可能世界上で、従来のラベリング法による論理検証を行えるようなアルゴリズムを提案する。具体的には、生成された複数の非決定性に対応するシミュレーションパスの集合、即ち「限定された可能世界」を、モデルチェッキング手法に準じた方法で論理検証できるような形で生成するためのアルゴリズムである。

((注 5.1)　上記(1), (2)の制約は、イベント記述文法(3 章)を使って表現可能である。)

なお、上記のような限定や近似を行って得られる「限定された可能世界」上での検証が妥当であるような対象の例として、バッチ式化学プラントがある。その理由は以下のとおりである。

・バッチ式化学プラントでは、ある処理の実行可能なタイミングが来たら、無意味な待ち時間を入れて無駄に待つようなことは行われず、直ちに実行を開始することが常である。そのため、上記(1)の「即時発火ルールの導入」は自然である。

・バッチプラントの場合、反応工程など、処理時間の遅延が起き易い箇所は自ずと限定されているため、上記(2)の「不確定性存在箇所の限定」は妥当な仮定である。

・バッチプラントでは、反応が予定の時間内に終了しなかった場合、その反応の完了を待って次の動作に移る。ただし、この時、その完了と同時に次の動作に移るのではなく、予め与えておいた幾つかのチェックポイントに合わせて次の動作に移らせるという操作方法を取ることも多い。つまり、本来連

続的な超過時間として発生する不確定性を、有限個のチェックポイントの選択という非決定性に置き換えて対応できるような運用方法が取られていることが多い。そのため、上記(3)の「不確定性の有限個の非決定性による集約近似」は、この種の分野では不自然な仮定ではない。

・これまでのバッチプラントのモデル化では、対象の規模の大きさからくる計算量の問題ゆえに、作ったモデルの妥当性のチェックは、幾つかの限られた設定値を与えての連続的な数値計算でのシミュレーション、(所謂、テストケースによる検証)によってしか行われていない 8)。そのため、モデルチェッキング法に準じて、論理的にこのような可能世界上で厳密な検証を行えるようにすることは、この分野では重要な課題となっている。

更に、本章の最後で、Japan Batch フォーラムが作成したバッチプラントの標準モデルへの適用実験を通し、上記のアプローチがこの種のシステムに有用であることを確かめる。

5.1 検証手法の枠組み

Fig.5.1 は ETSC に基づく本検証手法の枠組みである。基本的な考え方は次の通りである。まず、ETSC で対象システムの仕様記述を行う。次に、その ETSC を可能世界生成器により可到達木形式に変換する。ここで得られる可能世界とは、従来の時間 Kripke 構造を本研究での拡張事項であるイベントプールやジョブ識別子(IDJ)によって拡張した“拡張時間 Kripke 構造(：ETKS)”である(5.2 節)。一方、検証したい性質は時相論理 TCTL(:Timed Computational Tree Logic)で与える。そして、従来のモデルチェッキング手法で上述の可到達木(ETKS)が TCTL 形式を満たすかどうかを検証する。モデルチェッカの動作としては、TCTL のタイミング制約以外の部分には従来のラベリングアルゴリズムを適用する。また、TCTL のタイミング制約のチェックは、可能世界生成段階で導入するグローバルクロックの値を使って行う。

以下では、本稿での提案事項である ETKS の定義、及び、可能世界(ETKS)の作成手順を示す。

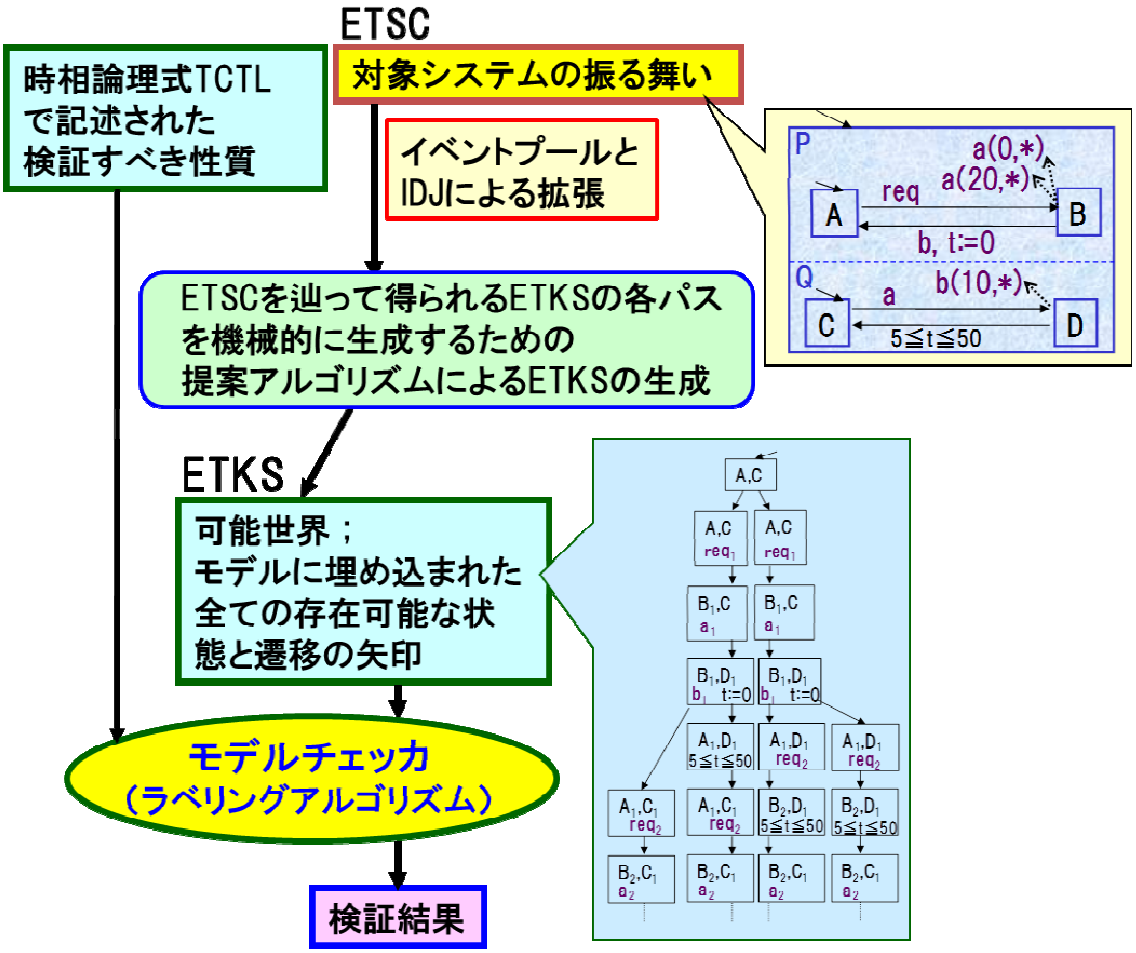


Fig.5.1　検証のための枠組み

5.2　時間 Kripke 構造の拡張

従来の時間 Kripke 構造(TKS)の 4 項目(N, P, R, π) に対し、イベントプールの内部データのダイナミクスを扱うための項目 P2 を追加導入した 5 項目 (N,P,P2,R,π)で拡張時間 Kripke 構造(ETKS)を定義した。

[定義 5.1] (拡張時間 Kripke 構造の構文)

・N:　ノードの有限集合

　AP:　原始論理式の集合

・P:　N 上の各ノードを、そのノード上で真となる全ての原始論理式の集合へ対応付けるラベリング関数

　P：N→2AP

・P2:　EP 上の各状態へ、EP 上で真となる原始論理式を対応付けるラベリング関数

　P2：SEP→2AP

・R:　ノード間の状態遷移

　R⊆N×N

・π：ノードをクロック変数へ対応付ける関数

　N→2C

5.3　「限定された可能世界」生成アルゴリズムの提案

冒頭で述べたような理由から、(1)不確定性存在箇所の限定、(2)不確定性の有限個の非決定性による集約近似、及び、(3)イベントの即時発火ルールの採用、により限定される非決定性の下で生成されるシミュレーションパスの全体､即ち､｢限定された可能世界｣の生成アルゴリズムを提案する｡ここで､(1),(2)は、イベント記述文法の XOR 型イベントを用い、明示的に指定し、(3)はイベントプールを使用して実現される。

本アルゴリズムの内容は、3 章で示した「EPM の基本操作」、及び「IDJ のための基本操作」を同時並行的に行いながら、「ETSC の動作」で示す方法で辿り得る全てのパスを持つ ETKS を生成するというものである。

以下、本アルゴリズムの具体的な手順を示す。

可能世界生成アルゴリズム

(準備) グローバルクロック(“Gc”とする) を一つ導入する。

(Step1)　ETKS のルートノードと初期 EP の生成

(1) ルートノードの生成：

まず､ETSC のルートの階層性･並行性を展開することにより､初期コンフィギュレーションを得る。それと共に、ETKS のルートノード(“N0”とする)を導入する。そして、当該ノード(N0)上に ETSC で得られた初期コンフィギュレーションの階層上最下位のロケーションをラベリングする｡(本稿では、以降も同様に、最下位ロケーションのみラベリングするとする。)

(2)　ルートノードに付随させる初期 EP の生成：ルートノード N0 に付随させる EP(“E0”とする) を導入し、予めそこへ初期イベントを登録しておく。具体的には、生産システムの場合、生産要求の発生を表すイベントをタイミング制約と IDJ 付きで登録する。

(Step2) ETKS の 後続ノードの生成

(1) ETSC のロケーションの遷移　：

現着目ノード(“Nk”とする) に付随する EP(“EPk”とする)に登録されているイベントのいずれか

が新たに発火可能となるまで Gc のカウンターを進める。発火可能となったイベント(“EVi”とする(“i”は IDJ))を ETSC へブロードキャストし、即時発火させる。以上の処理を、いずれかのロケーションの遷移が起きるまで繰り返す。ロケーションの遷移(“τ”とする)が生起したならば、先の(e3)に従って発火先のトークンに対する IDJ の処理を行い、(2A-1)または(2B-1)へ進む。

(2A-1) 後続ノードの生成(不確定性の指定がない個所):

“不確定性の指定がない”とは、即ち、ロケーションの遷移直後の生成イベントに XOR 型の指定がないことを意味する。この場合、ETKS の後続ノードは唯一つであるとし、当該後続ノード(“Nk+1”とする)を新たに導入するか、或いは、先に仮導入していた当該 Nk+1 の導入を確定的にする。そして、その Nk+1 に先ほどの発火イベント Evi をラベリングする。また、遷移τの生起で得られる新たなコンフィギュレーションをラベリングするために Nk+1 の次ノード(“Nk+2”とする)を仮導入し、遷移τの生起で得られたコンフィギュレーションの階層上最下位のロケーションを、当該時点でトークンが持つ IDJ を各々添えてノード(Nk+2)上にラベリングする。

(2A-2) 後続ノードに付随させる EP の生成

先行ノード Nk に付随する EPk に対し、次の操作を行うことにより、後続ノード Nk+1 に付随させる EPk+1 を得る。即ち、発火済みイベント Evi、及び、消費期限切れのイベントを抹消する。更に、ロケーションの遷移τの直後に生成イベントの指定(AND 型)があれば、その全てに対し、先の(e2)に従い、当該時点でトークンが持つ IDJ を添えて登録する。以上の操作の結果を内部に持つ EP を EPk+1 とする。

(2B-1) 後続ノードの生成(不確定性の記述がある個所):

“不確定性の記述がある”とは、ロケーションの遷移直後の生成イベントに XOR 型の指定があることを意味する。この場合、ETKS の後続ノードを、XOR 型で指定されている生成イベントの個数分導入する。

(後続ノードは“Nk+1-1,･･･,Nk+1-m”、XOR 型指定の生成イベントは“Ev1j,･･･,Evmj-”とする。ここで、IDJ として与える“j”は、先の(e2)より、当該イベントの生成元のトークンが持つ IDJ とする。)

もしくは、先に仮導入していた Nk+1 を XOR 型指定の生成イベントの個数に達するまで複製し、その各々に現ノード Nk からのパスを分岐させる。また、遷移τの生起で得られる新たなコンフィギュレーションをラベリングするため、“Nk+1-1,･･･,Nk+1-m”の次ノード(“Nk+2-1,･･･,Nk+2-m”とする)を仮導入し、遷移τの生起で得られたコンフィギュレーションの階層上最下位のロケーションを、当該時点でトークンが持つ IDJ を各々添えてノード Nk+2-1,･･･,Nk+2-m 上に各々ラベリングする。

(2B-2) 後続ノードに付随させる EP の生成

後続ノード Nk+1-1,･･･,Nk+1-m の各々にイベントプール EPk+1-1,･･･,EPk+1-m を独立に与える。各 EPk+1-1,･･･,EPk+1-m の各々の内容は、次の操作を行うことにより確定する。即ち、まず、EPk+1-1,･･･,EPk+1-m の内部に、EPk の内容から、発火済みイベント Evi、及び、消費期限切れのイベントを抹消したものを与える。ここまでは、AND 型指定の場合と同じである。AND 型と異なるのは、生成イベントの登録の仕方である。具体的には、EPk+1-1,･･･,EPk+1-m の各々に、XOR 型指定の生成イベント Ev1j,･･･,Evmj を別々に 1 つずつ登録していく。(つまり、EPk+1-1 に Ev1j、EPk+1-2 に Ev2j,･･･となる。)

以降、各終端ノードを着目ノードとし、(Step2) 以降の処理を行う。EP 内が空となり、コンフィギュレーションが変化しなくなった時、実行は終了する。

[例 5.1] (アルゴリズムに基づく可能世界の生成)

Fig.5.2 は、提案するアルゴリズムに従って、図中の ETSC を ETKS へ変換した結果を示している。変換の過程は次の通りである。

(Step1)ETKS のルートノードと初期 EP の生成

(1)まず、ETSC の初期コンフィギュレーション{P,Q,A,C}を得ると共に、ETKS のルートノード(N0)を導入し、ノード N0 に{A,C}をラベリングする。

(2)また、ノード N0 に付随させてイベントプール(E0)を導入し、そこへ初期イベントとして“req1(10,*)”,”req2(40,*)”を登録する。ここで、添え字の 1，2 は IDJ である。

(Step2) N0 の後続ノードの生成

(1) Gc のカウンターを 10 進めると、E0 に登録されているイベントの一つ“req1”が発火可能となり、“req1”を ETSC へブロードキャストする。このとき、ETSC ではロケーション A から B への遷移が可能となるため、 即時発火で、req1 を発火させながら当該遷移を生起させる。更に、req1 の発火はジョブ切り替えタイミング(即ち、オートマトンの初期状態からの遷移を伴うイベント発火)であるため、先の (e3)に従い、req1 の発火先であるロケーション P 上のトークンに IDJ“1”を与える。

(2B-1) 後続ノードの生成 (不確定性の記述がある個所) :

ロケーションの遷移 A→B の直後の生成イベント a(0,*) と a(20,*)の間に XOR 型指定があることから、N0 の後続ノードを 2 個 (“N1”と“N2”)導入する。そして、その各々に対し、現ノード N0 からのパスを分岐させて与える。また、N1,N2 の次ノードとして N3,N4 を仮導入し、遷移 A→B の生起

で得られたコンフィギュレーションの階層上最下位のロケーション(B,C)を N1,N2 の各々にラベリングする。ただし、ロケーション B は P 上のロケーションであり、P 上のトークンは IDJ “1” を持つ。従って、B は IDJ を添えた “B1” としてラベリングする。

(2B-2) 後続ノードに付随させる EP の生成

後続ノード N1,N2 の各々にイベントプール E1,E2 を独立に与える。E1,E2 の内容は、次の操作を行うことにより確定する。即ち、まず、E1,E2 の内部に、E0 の内容から、発火済みイベント req1 を抹消したものを与える。次に、E1,E2 の各々に、XOR 型指定の生成イベント a(0,*),a(20,*)に、当該生成元 P のトークンが持つ IDJ “1” を付加した a1(0,*),a1(20,*)を、別々に 1 つずつ登録する。つまり、E1 に a1(0,*)、E2 に a1(20,*)を登録する。

(以下省略)

以降は、 N1 の後続ノード、及び、N2 の後続ノードは、 (Step2)以降の手順に従って、各々別々に生成していく。

Fig.5.2 の ETKS は、上記方法で得られた可能世界全体である。この ETKS では、本稿での提案事項である IDJ(ここでは、IDJ=1,2)により、ジョブごとに実行パスが辿られるようになっている。また、XOR 型生成イベントで限定的かつ離散的に指定した不確定性の箇所と内容は、分岐パスの箇所と内容として、各々実現できている。なお、各 EP に付した “gt” は、グローバルクロックの値を示している。

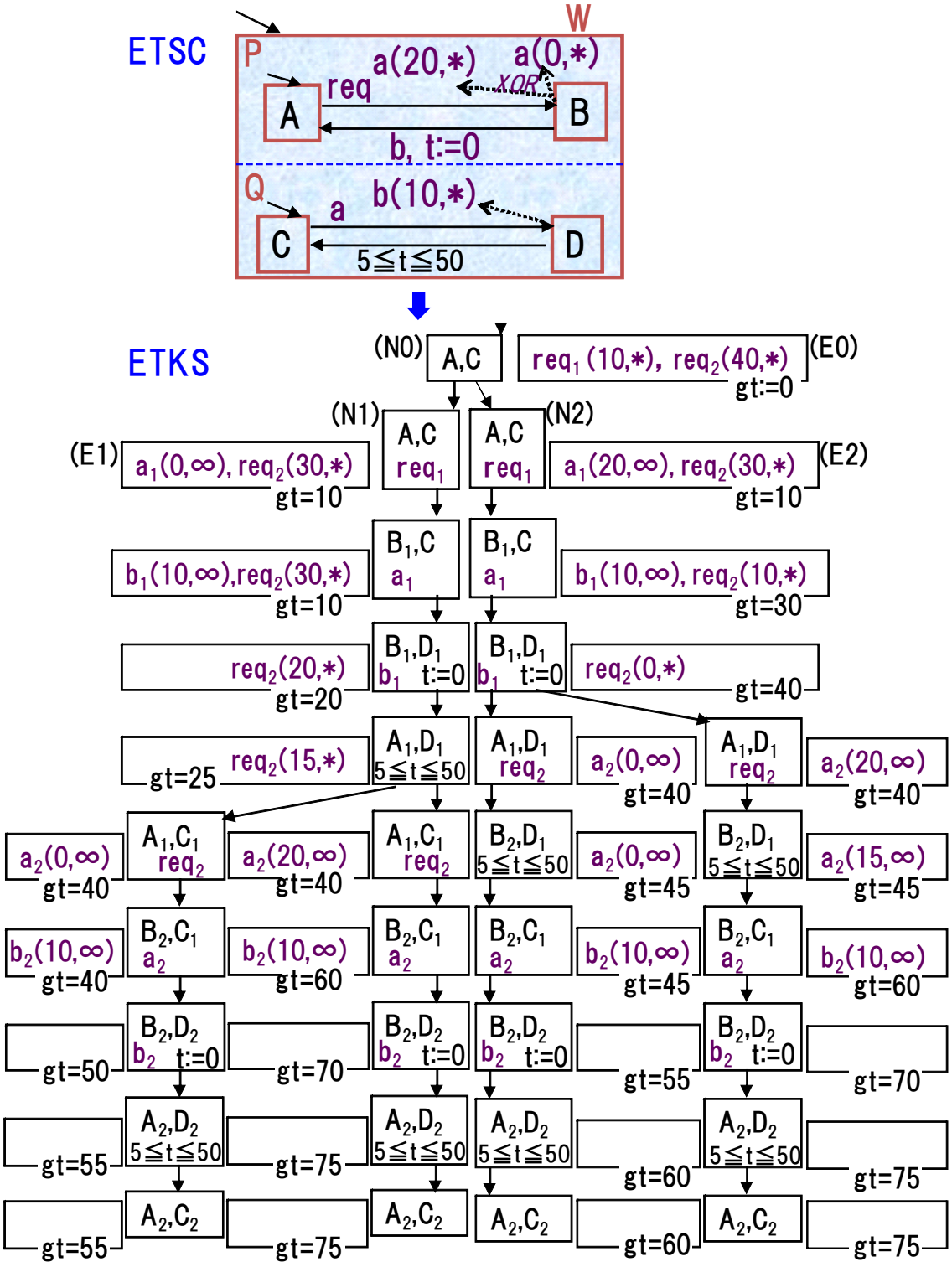


Fig.5.2: 可能世界生成アルゴリズムの適用例

5.4 検証手法の実装

本研究で実装した検証手法の枠組みは Fig.5.1 で示した通りである。ここで、先に述べたように、可能世界生成器として 5.3 節の「限定された可能世界」生成アルゴリズムを実装した。また、モデルチェッカとしては、基本的に通常のラベリングアルゴリズムを使用した。ただし、タイミング制約の判定は、可能世界生成で用いるグローバルクロックの値(Fig.5.2 の各“gt”)を利用すると容易に行えるため、グローバルクロック値に基づく判定方法を導入し実装した。

5.5 提案手法が適応可能な問題領域

以上のように提案する方法により生成される可能性世界による検証が妥当であるような対象の一般的な性質・条件は次の通りである。

・処理の実行可能なタイミングが来たら、特に理由がない限り、直ちに実行を開始されること。

・処理時間の不確定性の発生する箇所が計算可能なレベルに限定されていること。

・処理時間の不確定性がシステム全体に及ぼす影響を検証する場合、その不確定性を有限個の非決定性に抑えてチェックできるような運用方法が取られていること。

5.6　適用例

以下では、Fig.5.3 で示す JBF 作成の標準的バッチ式化学プラント[25]を対象として、そこに与えられた運用操作の妥当性(健全性，制御可能性)の検証が本稿で提案した挙動モデルと検証アルゴリズムを用いて行えることを示す。このことは、上に示したモデルとアルゴリズムによるアプローチの妥当性と有効性を間接的に示すものである。

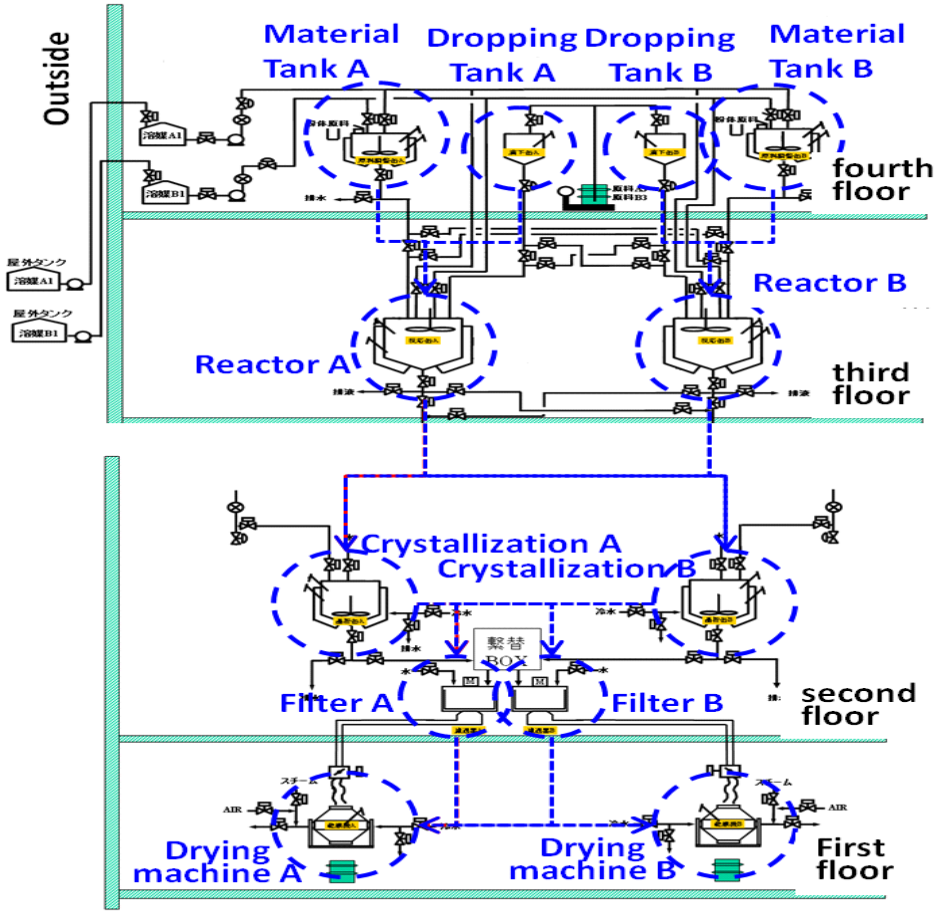


Fig.5.3 対象プラント（JBF 作成の標準モデル）

5.6.1　バッチプラント設計上の問題の所在（- 検証されるべき事柄は何なのか？ -）

冒頭でも述べたように、変種変量生産が求められるバッチプラント生産系では、最適スケジュールに従って運用が行われるが、時間ベースのスケジューリング結果だと、不確定な変動に対応できない。そのため、スケジューリング結果が持つ最適性は可能な限り保ちながら、小さな不確定性は一定の効率低下の範囲内で吸収できるような“ロバストな制御”が行えるような事象駆動での制御に変換しなくては

ならない。当該変換を行う方法として、本研究では、順序情報を使用する方法を開発した 1)。具体的には、スケジューリングで与えられている「ある装置上での処理開始イベントの発生順序」を取得し、当該順序情報を取り込んだオートマトンを生成する方法を開発した。しかし、不確定要因に因る時間遅れが大きくなり過ぎると、当初のスケジュール通りに作業を進めること自体に意味がなくなるため、上述のようなロバストな制御すら役に立たなくなるケース(以降、“例外時”と記す) がある。以上のことから、バッチプラントの制御系としては、最適スケジュールベースで作成すべき通常の制御方法と、例外時対応のための制御方法との双方を併せ持つような制御系を構築する必要がある。更に、以上のような 2 種類の制御方法を持つ制御系の場合、制御方法の内容のみならず、何時、どのような方法で、通常制御から例外時制御へ切り替えさせるか、といったことも含めて設計事項となる。

以上のことから、本実験における「制御系の妥当性検証」で検証されるべき項目とは、当該制御系が持つ通常時の制御方法、例外時の制御方法、通常時制御から例外時制御への切り替えのタイミング、及び、その切り替え方法となる。

5.6.2 対象プラント

対象プラント概要

本プラントは A,B 2 系列の装置構造を持つ。また、X と Y の 2 種類の製品製造が可能である。製品製造の流れは、複数の原料から混合物を生成し、混合物と他の原料を反応させ、結晶化処理を行い、結晶を濾過乾燥させ、製造終了という流れである。更に、反応缶以降では、A,B 系間でのたすきがけ運転が可能である。また、実行中の装置間での中間原料移送のための待機可能時間は各装置とも 150 である。

不確定性の所在

本実験では、性質検証の結果に重大な影響を及ぼす可能性のある不確定性として、乾燥機 A, B での処理時間の不確定性のみを限定的に扱う。具体的に、各系各製品に関する処理終了チェックのタイミングは次の通りとした。;

(A 系，製品 X)→200 と 500、(A 系，製品 Y)→300 と 600、

(B 系，製品 X)→300 と 600、(A 系，製品 Y)→200 と 300

ここで、「(A 系，製品 X)→200 と 500」の意味は、「A 系列で製品 X に関する乾燥処理にかかる時間は 200 以上 500 以下（標準で 200，遅くとも 500)」となる。

制御方法

[通常時の制御]： Fig.5.4-(1)のようなガントチャートが示すスケジューリング結果を使用する。ここ

で、ガントチャートの各セグメントの長さは、各処理の定常時の所要時間を表している。具体的な使用方法は次の通りとした。

(1) まず、制御すべき事柄を明らかにする。ここでは、たすきがけ運転が可能な反応缶以降の各装置において、「処理終了後の中間原料の移送先は A 系、B 系どちらの装置か？」 ということが制御すべき事柄である。

(2) 上述の“制御すべき事柄”に対する制御手順を、Fig.5.4-(1)から、「ある装置上での処理開始イベントの発生順序」という形で取り出す。具体的に、移送先装置が晶析缶(A または B)のケースで説明すると以下の通りとなる。;

① 晶析缶 A,B 上の「移送受入開始タイミング」間の全順序関係を取り出す。(Fig.5.4-(2))

② ①で取り出した移送先(A／B)に関する順序情報の各要素に対し、必要な制御情報をガントチャートから取り出し添付する。ここでは、移送対象を表す製品タグ(即ち IDJ)を添付する。(Fig.5.4-(3))

[例外時の制御]： 本実験では、A 系装置優先のディスパッチングルールを使用した。ここで、通常時から例外時への制御の切り替えタイミングは、各装置の待機可能時間が”150”であることを考慮し、次装置への移送待ち時間が 100 を超えた時点とした。また、切り替え方法は次の通りとした。;

(1) ある装置での処理終了時に、当該装置が持つ中間原料の移送先となる装置への同期メッセージを出力する。この時、待機待ち時間が 100 を超えた時点で発火可能となるイベント(以降、“切り替えメッセージ”)も併せて出力する。

(2) 同期メッセージに対する応答が、待ち時間“100”を超える前に得られる場合、予定通り移送処理を行い、以降も、通常時の制御に基づく実行を続ける。

(2’) 同期メッセージに対する応答がないまま応答待ち時間“100”が経過する場合、当該タイミングで発火可能となる上記の“切り替えメッセージ”を発火させる。この時、同時に例外時処理のモードへ処理系を移行させる。以降、例外時制御に基づく実行を最後まで続ける。

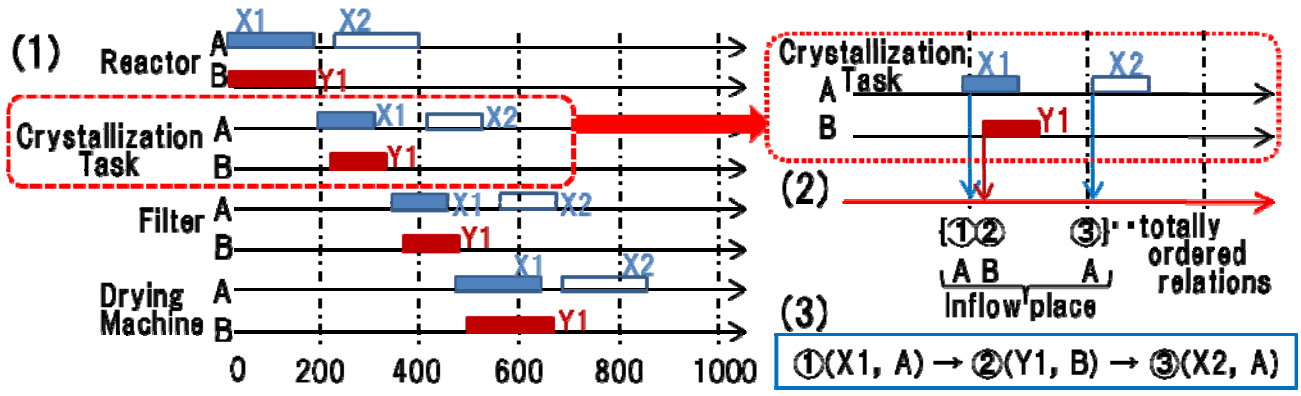


Fig.5.4: 生産スケジュールに関するガントチャートの例

5.6.3　対象システムのモデル化

本対象プラントは、装置の振る舞いを表す“装置オートマトン” 12 個と、制御方法を表す“制御オートマトン”4 個 、都合 16 個のオートマトンにより記述できた。その概観は Fig.5.5 のようになる。また、Fig.5.6 は乾燥器に関する装置オートマトンの一部であり、図中”dry_x(100,*) XOR dry_x(400,*)”は、不確定性を持つ当該装置の乾燥終了をチェックするタイミングの記述である。また、Fig.5.7 は、反応缶に関する装置オートマトンの一部だが、図中、「制御モード切り替えのための仕組み」の記述“sync_x{ from 反応缶 A,to 晶析缶 A}(1,*) AND timeLimit_x{ 反応缶 A}(100, *)” を持つ。即ち、同期メッセージと例外処理切り替えタイミングを知らせるためのメッセージを同時に出力させるための記述である。(詳細は Appendix-A2.1)

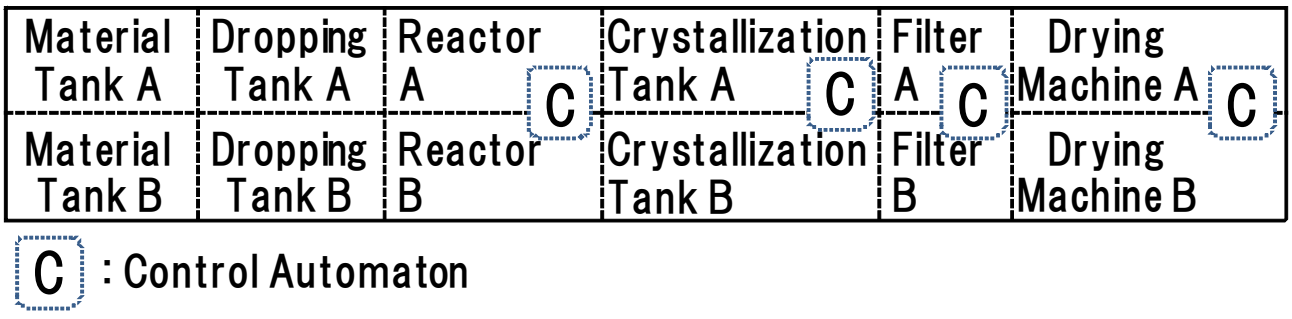


Fig.5.5:　対象プラントの挙動表現の階層構造上最上位の ETSC モデル

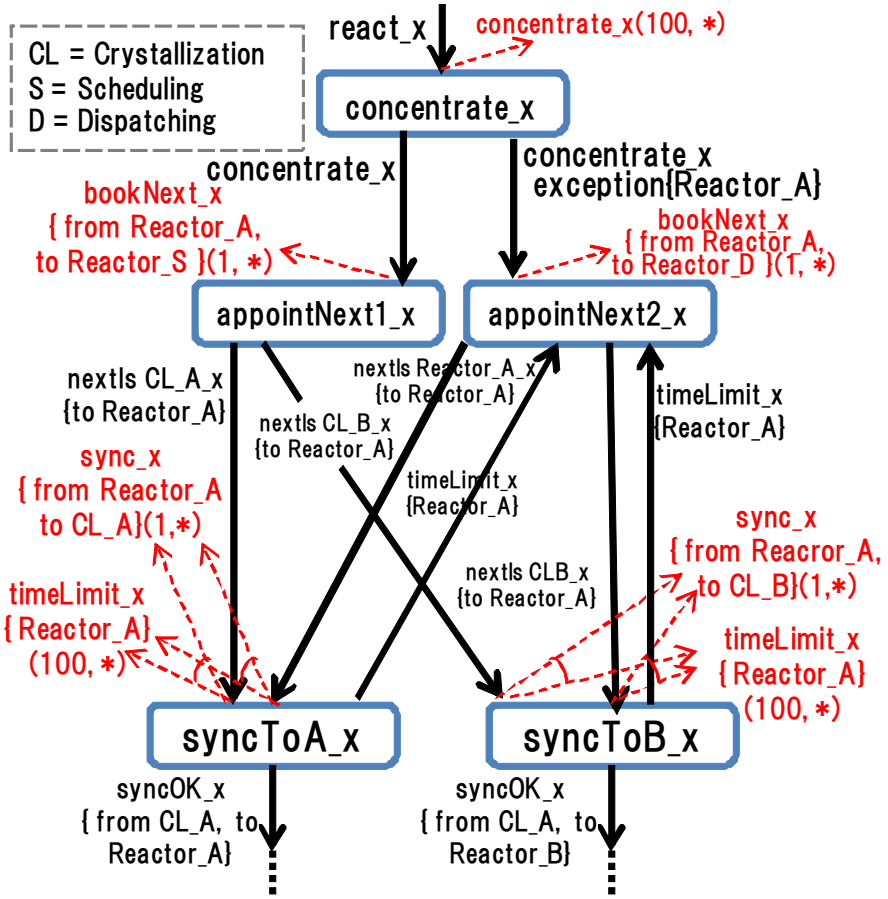


Fig.5.6: 乾燥器周辺のサブシステムの挙動表現モデル

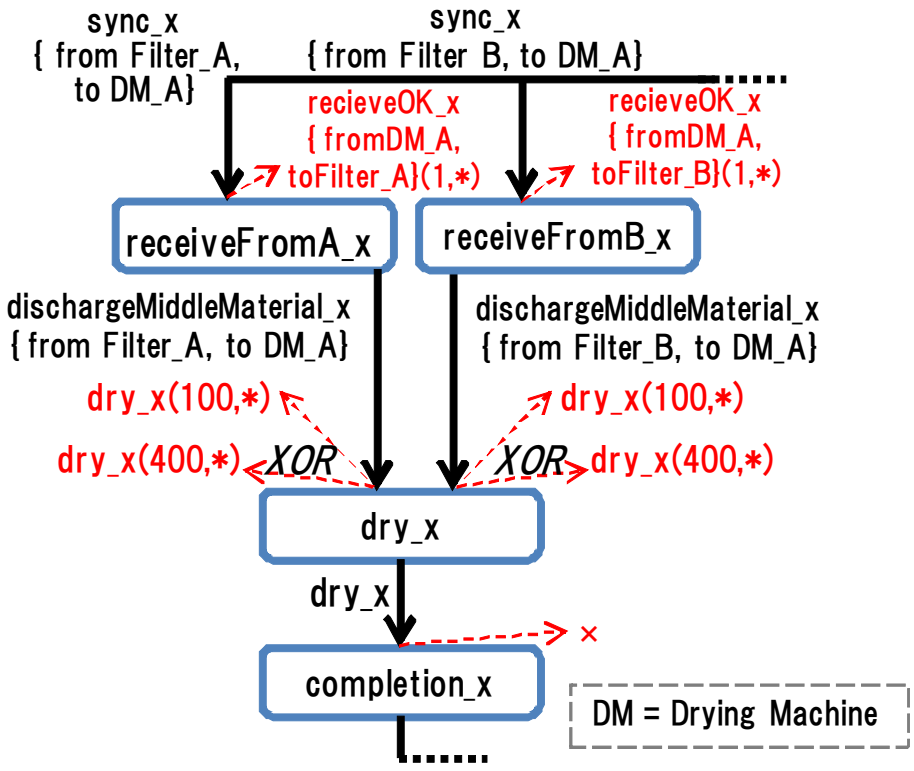


Fig.5.7: 反応器周辺のサブシステムの挙動表現モデル

5.6.4 検証実験と結果

[実験 1]：通常制御から例外時制御への“切替え方法”と“切り替えタイミング”の妥当性チェックができることを確認した。具体的には、まず、5.6.2 節で示した通常制御、例外時のための制御、制御切り替えタイミング、及び切り替え方法に基づき制御系を設計した。そして、当該制御系を持たせる形で対象プラントの挙動表現モデルを作成、及び実行し、実行結果をトレースした。その結果、当該切り替えが、指定した方法及びタイミングで正しく実現されていることが確認できた。

[実験 2]：システムの適応限界のチェックが可能であることを確認した。具体的には、処理時間に不確定性のある 4 箇所(5.6.2 節)のうち、「乾燥機 A の製品 X の乾燥終了」をチェックするタイミングを表すパラメータのみを変更しながら、各ジョブの (f1)総処理所要時間及び、(f2)中間原料待機時間に関するタイミング制約の充足可能性を以下の論理式を使って検証した。

(f1) 総処理所要時間に関する検証のための論理式

AG(event(req[X-1]) → AU(time1500 , OR(state({DR_A}END[X-1]) , state({DR_B}END[X-1]))))

(f2) 待機時間に関する検証のための論理式

event(sync{FI_A - DR_A[X-1]}) → AU(difftime150 ,event(receiveOK{DR_A-FI_A[X1]})

ここで、論理式中の[X-1]の“1”は IDJ を表している。本実験では、IDJ の値、即ち、着目するジョブを変えながら、上記検証を行った。以上のような検証の結果、乾燥機 A の製品 X の処理時間の適応限界は 313 時間であることが確認できた。

[実験 3]： 例外時用制御への“切り替えの効果”がチェック可能であることを確認した。具体的には、切り替え“有り”と“無し”の各々のケースで得られる可能世界を、IDJ をキーとしてジョブごとにトレースした。そして、グローバルクロックの値から各々のケースでの総所要時間及び中間原料待機時間を算出し比較した。その結果、双方の所要時間とも、“切り替え有り”のケースの方が小さくなると分かった。以上のことから、制御切り替え効果のチェックが可能であることが確認できた。

以上の実験を通し、本提案手法の妥当性・有用性が確認できた。なお、以上のような実験は、マシン：HP Compaq dc7900、プログラミング言語 : Java(jdk-6u24-windows) 、OS : Windows7(64bit)、CPU のクロック数：2.93GHz、という実行環境下において、Table5.1 に示す規模での検証が実行可能であった。

Table 5.1:　計算可能なシステムの規模と実行にかかる時間

	The number of batches	The number of nodes of ETKS (nodes)×(paths)	Computation time
Schedule	10	1400 × 1024	10sec
Priority rule	8	1900 × 256	8sec
Schedule + Priority rule	7	1600 × 128	8sec

Exceptional handling

5.6.5　実験の考察

・本実験での実行結果のトレースより、複数ジョブが同時並列に処理され、制御方法の通常時から例外時への切り替えもあるような複雑な対象システムの振る舞いが、EPM によるイベント操作により、簡潔に扱えることが確認できた。(実験 1)

・計算量爆発回避のために導入した３つの限定や近似、即ち、(1)不確定性存在箇所の限定、(2)不確定性の有限個の非決定性による集約近似、(3)イベントプール上での即時発火ルール使用は、バッチプラント設計上の問題を扱う上では妥当であることが確認できた。(実験 2)

・IDJ 付き可能世界上での動作検証の結果、システムの適応限界、及び、制御ルール切り替え効果をジョブごとにきめ細かくチェックできたことから、IDJ の有用性が確認できた。(実験 2，3)

・EPM によるイベント操作の導入により、個々のイベントの発火可能期間が本質的に重要な制約条件となる生産システムの振る舞いが、扱えるようになったことが確認できた。

5.7　5 章のまとめ

本章では、3 章で提案した拡張時間ステートチャート(既存の時間ステートチャートを「EPM に基づくイベント管理方法」や「ジョブ識別子(IDJ)」で拡張したモデル)のための動作検証を、従来のモデルチェッキング手法で行えるようにするため、次のことを行った。即ち、この種の検証手法が持つ可到達木生成段階での計算量爆発の問題への一対処方法を考案し、それが有効活用できる対象領域について示した。そして、Japan Batch フォーラム で提案される標準的バッチプラントを適用例とし、本手法の有効性を確認した。

第 6 章　結論

本論文では、多数の逐次操作や並列操作が階層的かつ事象駆動で実行されている離散型並列生産システムの複雑な動作を明快に表現できる挙動表現モデルを開発し、次にそれに基づいて「不確定性のある離散型生産システムの系統的制御系設計法」を提案した。また、この挙動モデルを用いて、実システム稼働時に与えられる運用戦略や制御則の妥当性検証を、モデルチェッキング型検証法の枠組みで行う方法を与えた。具体的には以下の通りである。

（1）離散型並列生産システムの挙動表現モデル ETSC の提案　(3 章)

離散型並列生産システムの「動作的特徴」を列挙し、これら動作的特徴を明確に扱えるような挙動表現モデルを開発した。開発の仕方としては、離散型並列生産システムのモデル化に必要な要件を元々最も多く備えている既存モデル「時間ステートチャート」(TSC)をベースモデルとして導入し、その TSC で扱えない「動作的特徴」(複数ジョブの並列実行、制御方法の例外時の切り替え等)を扱えるように TSC を拡張するという方法をとった。具体的には、イベントの生成、発火可能期間、及び、ジョブ-イベント間の関係性を持たせるための“ジョブの ID”が扱えるように TSC を拡張した。

以上の拡張の結果得られた挙動表現モデル「拡張時間ステートチャート」(ETSC)を用いることにより、離散型並列生産システムの複雑な振る舞いを明確に表現できるようになった。なお、得られた ETSC は、これまで人手を介して行われてきた離散型並列生産システムの制御系設計を、モデルベースでかつシステマティックに行うためのベースモデルとして利用可能であることが分かった。更に、ETSC は、これまでテストデータを使用したシミュレーション程度の検証しかできなかったような実用規模の生産プラントの動作を、有る程度厳密かつ論理的な方法で検証するためのモデルとしても使用可能であることが分かった。

（2）ETSC をベースモデルとする「制御系設計のためのフレームワーク」の提案 (4 章)

不確定性を有する離散型並列生産システムの制御系設計を、モデルベースで、かつシステマティックに行えるようなフレームワークを開発した。

具体的には、まず、記述上の個人差を無くすために ETSC を限定したサブセット「R-ETSC」を導入することにより、モデルベースでの分かり易い制御系設計が可能となった。なお、このような限定は、生産システムが本来持つ構造的特徴を直に表現可能なプロセスネットワーク型モデルを中間モデルとして導入することにより形式的な手順で行えるようになった。また、このプロセスネッ

トワーク型モデルを活用し、何らかの制御を要する箇所を、ネットワーク構造の図形的特徴から機械的に特定できるようにした。

次に、計画系で時間駆動の情報として得られる目標スケジュールを、処理時間が多少変動しても利用可能な事象駆動の情報へ、系統的に受け渡すことを可能にした。具体的には、まず、制御に関する情報を、スケジュールによって動的に変わる部分と、スケジュールによって変わらない静的な部分とに分けて記述できるような 5 種類のテンプレートを導入した。また、これらテンプレートに埋め込む情報を、R-ETSC からシステマティックな手順で抽出する方法を提案した。更に、以上のようなテンプレートを利用する方法を使って、より大きな外乱により再スケジューリングが不可避となった場合においても、現在稼働中の対象プラントを止めることなく、その再スケジューリング結果を取り込み、スムーズにシステム運用を継続できるような技術的方法を導入した。

本アプローチの妥当性と有効性は、スケジュールの動的変更を自動的に行わせるためのプログラムを実装し、FA システムを適用例とした検証実験を通して確認した。

以上のような提案手法を用いることにより、これまで、各企業が独自に蓄積してきたノウハウに基づき行ってきた制御系の設計、及び、適宜、差立や、制御プログラムの追加・変更など、かなりの手作業を介して行ってきたスケジュールの変更を、モデルベースで分かり易く、かつシステマティックな方法で行えるようになった。

(3) ETSC に基づく動作検証手法の開発 (5 章)

従来の TSC で利用可能なモデルチェッキング型の論理検証の手法に対し、ETSC でモデル化した現実規模の生産プラントに適用できるようにするための拡張を行った。具体的には、まず、既存の手法、即ち、Kripke 構造全体を時間領域分割して状態ノードを作成しその上での検証を行う検証手法が元々持っている計算量爆発の問題への対処方法を導入した。即ち、対象プラントの可能な全ての振る舞い、即ち、“可能世界” に対し、(1)不確定性存在箇所の限定、(2)不確定性の有限個の非決定性への置き換えによる集約近似、(3)イベントプール上での即時発火ルールの採用、という３つの制約を与えた。そして、それによって限定生成された可能世界上で、従来のラベリング法による論理検証を行えるようなアルゴリズムを提案した。

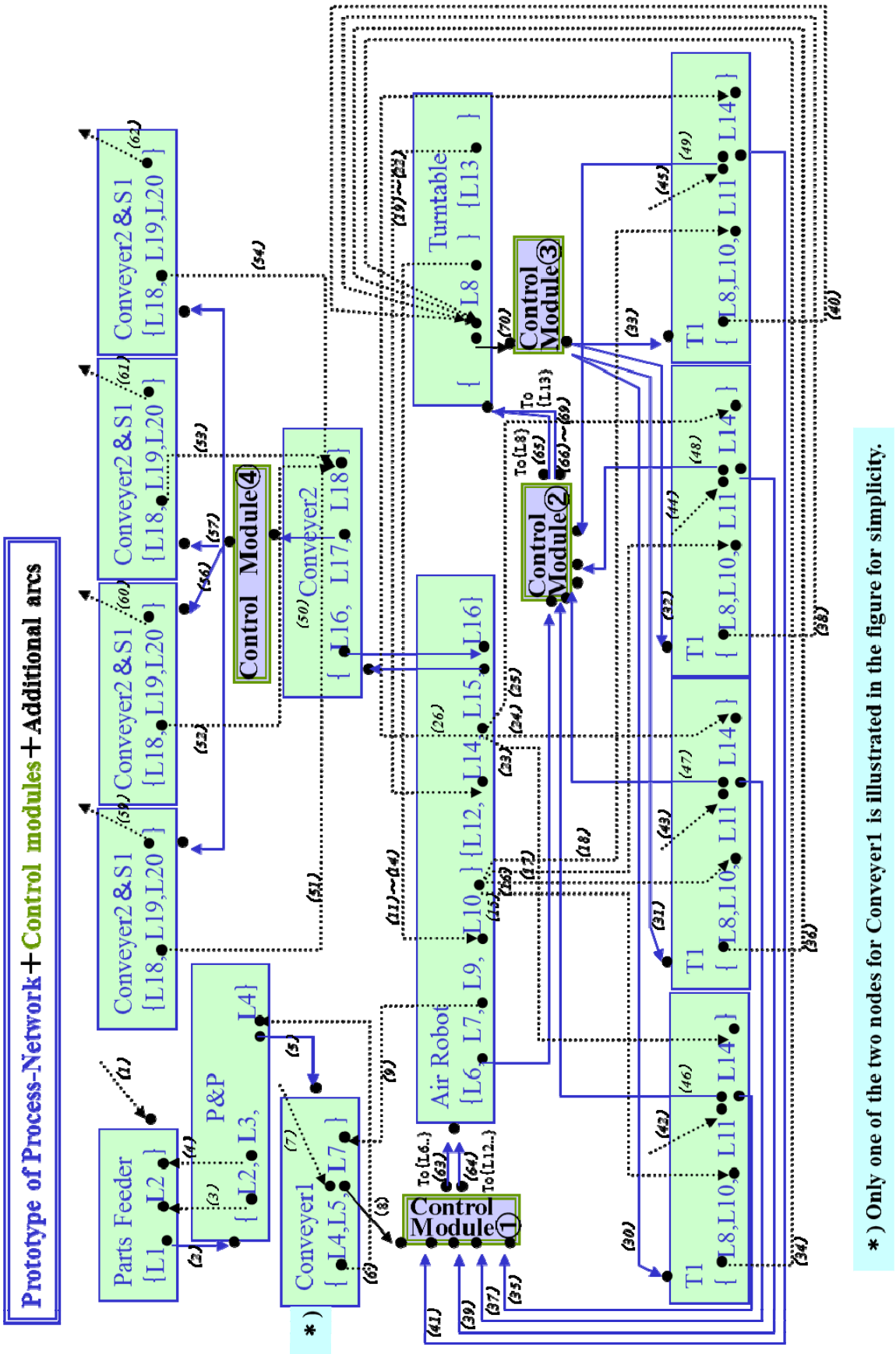
本アプローチの有効性は、限定や近似を行って得られる「限定された可能世界」上での検証が妥当であるような対象の例としてバッチ式化学プラントを導入し、これを適用例とした検証実験を通してその有効性を確認した。

以上のような提案手法を用いることにより、これまでテストデータを使ったシミュレーション程度のチェックしか行われてこなかった実用規模のバッチ式化学プラントの動作検証が、モデルチェッキング手法に準じた有る程度厳密かつ論理的なやり方で検証できるようになった。

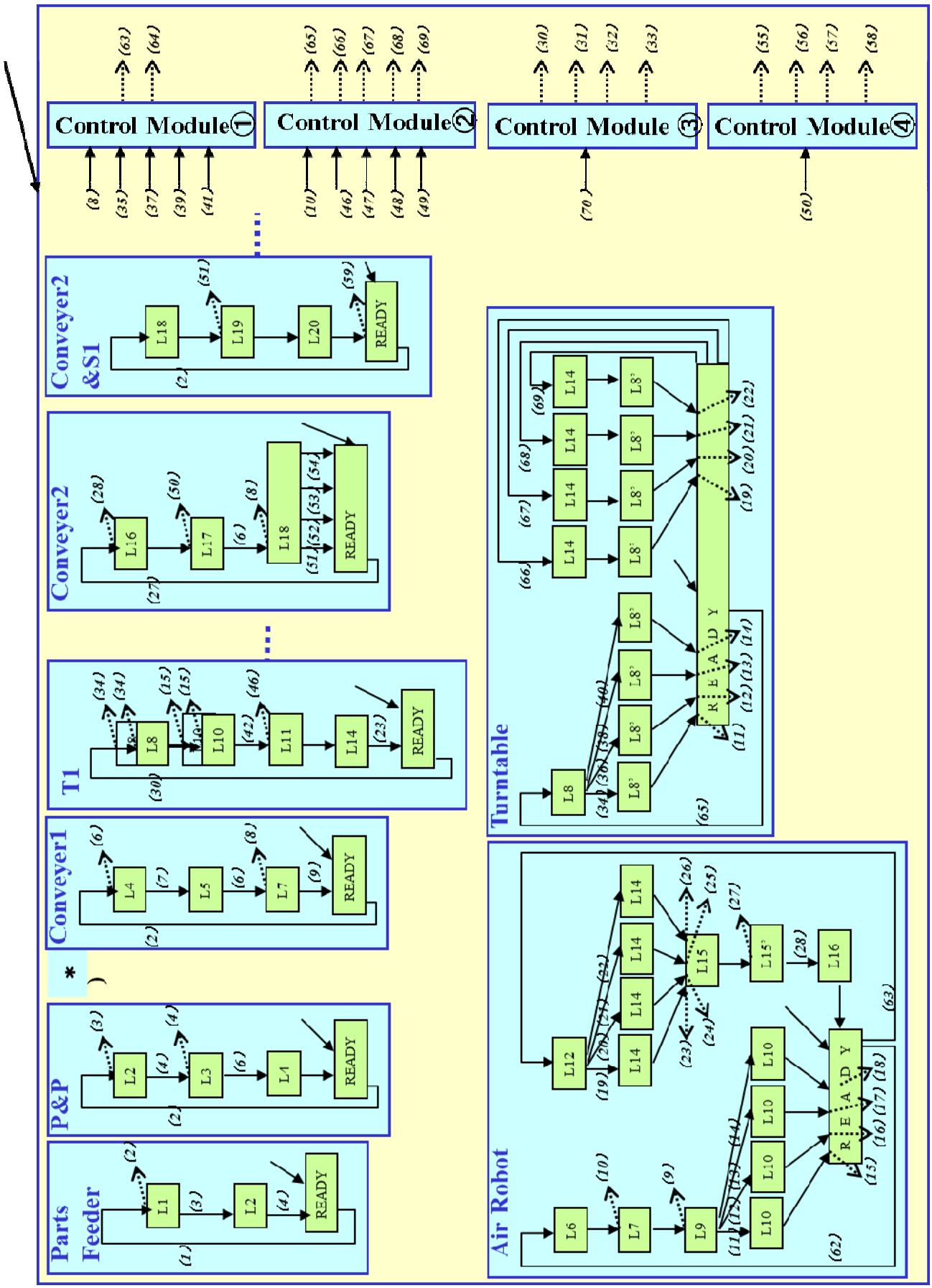
以上が本論文のまとめである。なお、残された課題としては、本手法は離散的な手法であるため、連続的な遅れの影響はチェックできないということである。また、計算量の問題ゆえ、複数個所の不確定性の組み合わせの影響が問題になるケースを扱うことが困難である。そこで、複数個所の連続的な遅れの影響を評価する方法に関する検討が今後必要と考えられる。また、本論文において、拡張時間ステートチャート(ETSC)は、離散型並列生産システム固有の複雑さを表現できるようなモデルとして開発した。しかし、その表現力の高さゆえ、生産システム以外の離散事象システムの挙動表現モデルとしての利用可能性も十分あると考えられる。そこで、今後は、生産システム以外の対象に関する問題解決にETSC モデルを有効活用する方法についても検討していく予定である。

Appendix

A1.1　FA 実験装置のプロセスネットワーク型モデル (骨格部分全体)

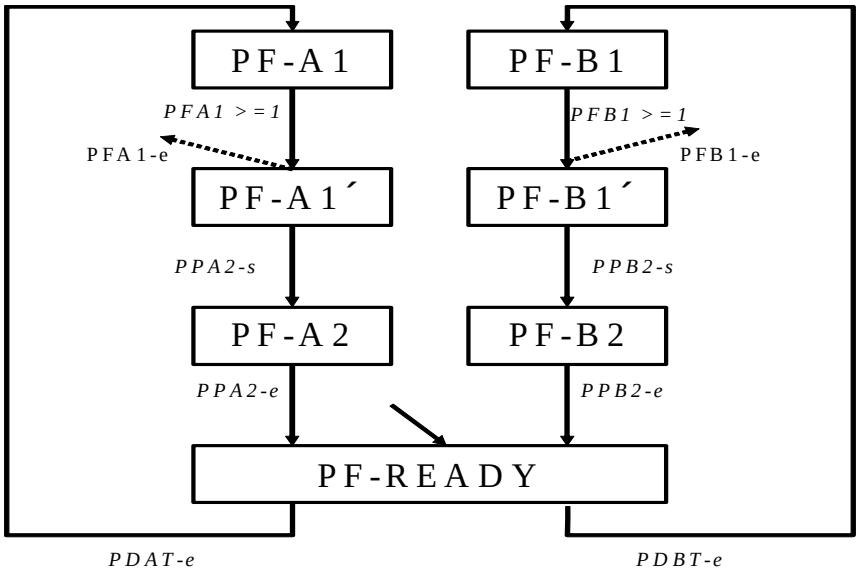


A1.2　FA 実験装置の ETSC モデル　(全体像)

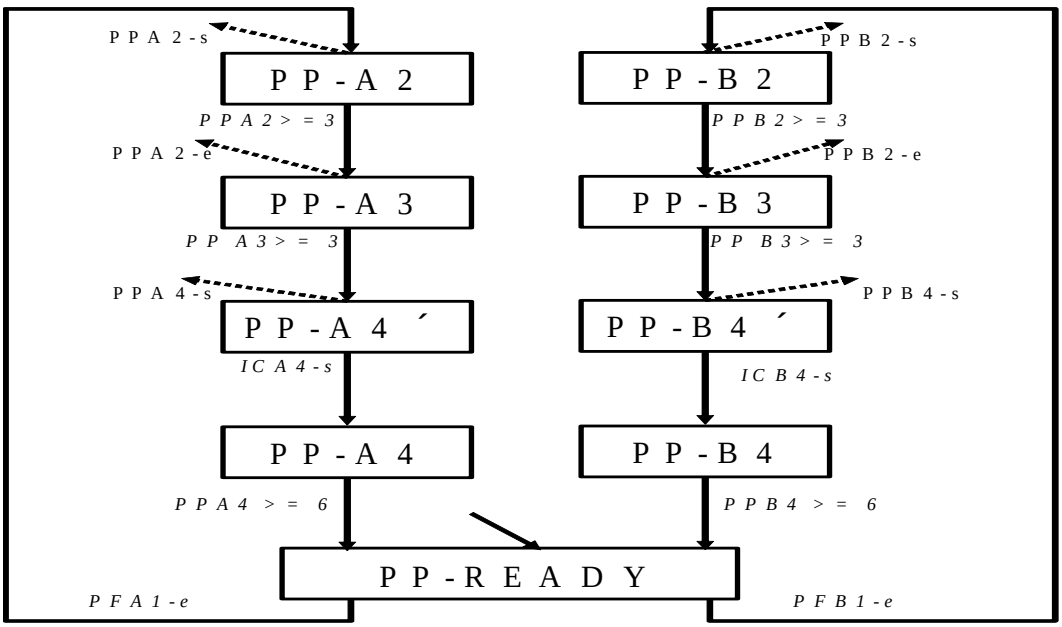


A1.3　ETSC モデルの詳細

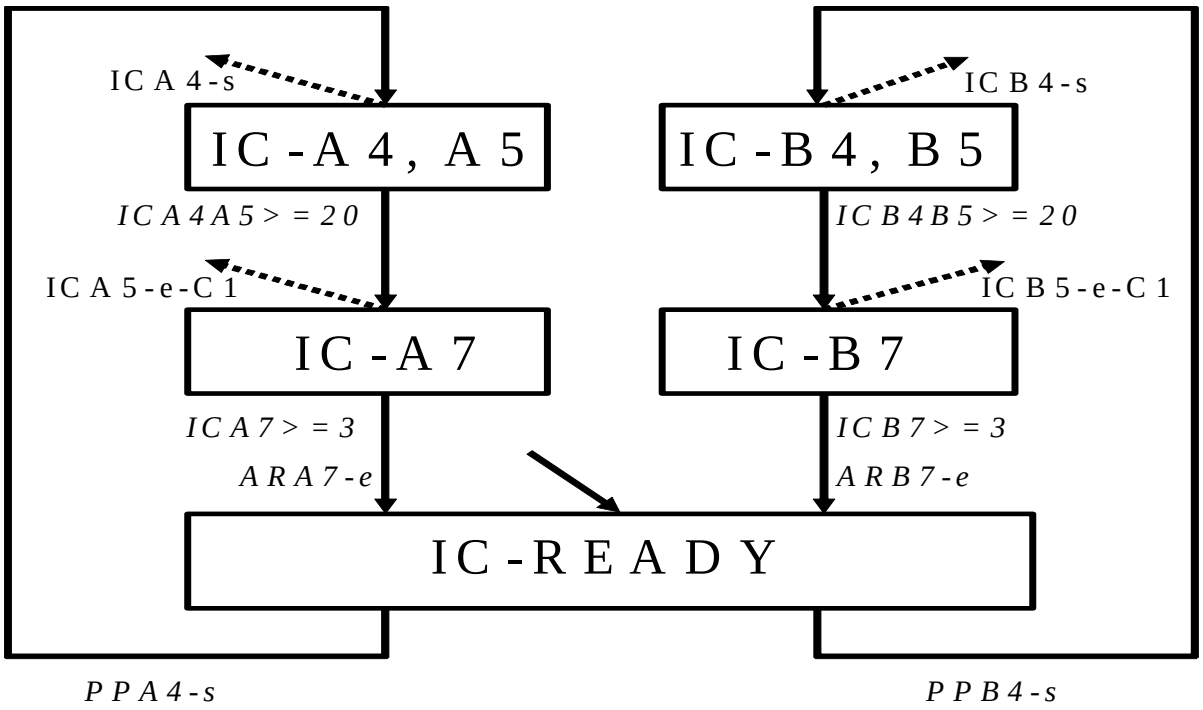
パーツフィーダ



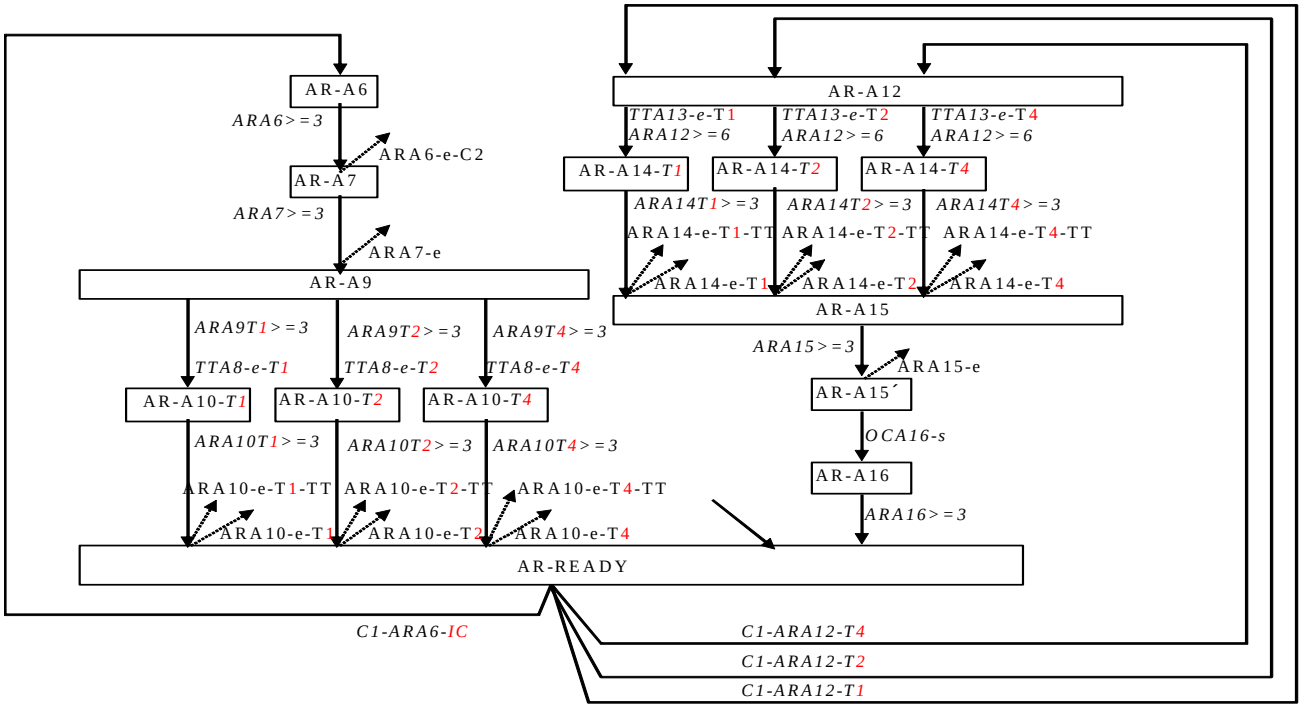
搬入コンベア



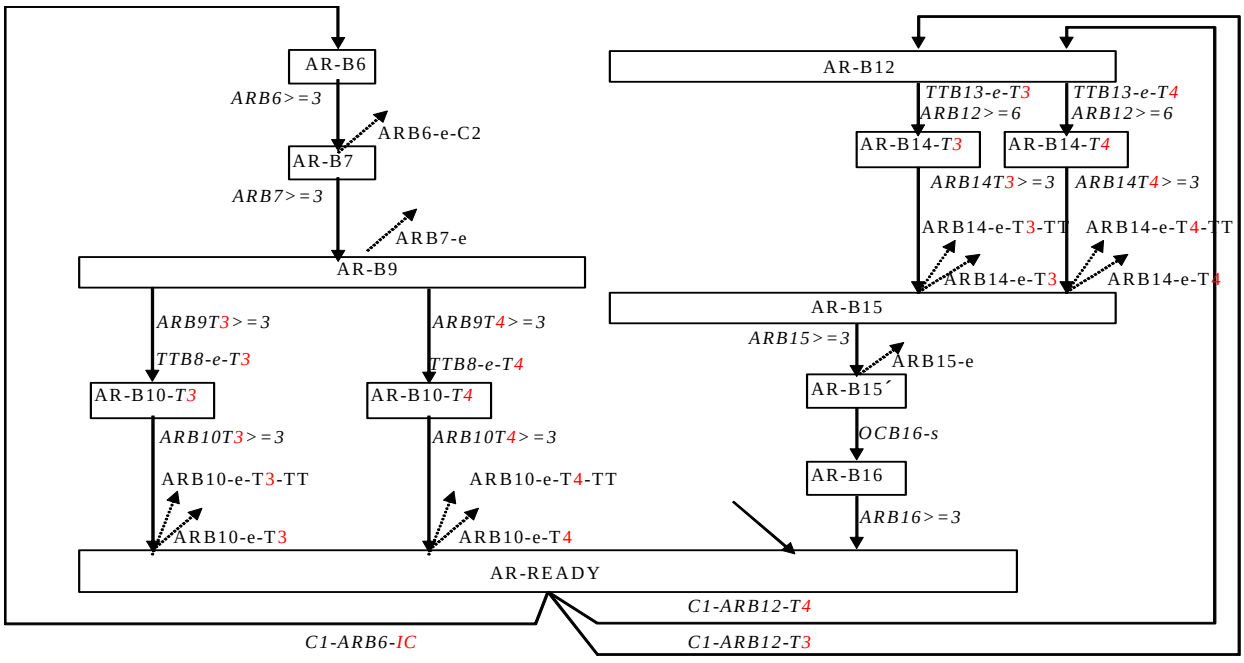
ピック＆プレース


IC A 4-s
IC B 4-s
IC-A 4, A 5
IC-B 4, B 5
ICA4A5>=20
ICB4B5>=20
IC A 5-e-C 1
IC B 5-e-C 1
IC-A 7
IC-B 7
ICA7>=3
ICB7>=3
ARA7-e
ARB7-e
IC-READY
PPA4-s
PPB4-s

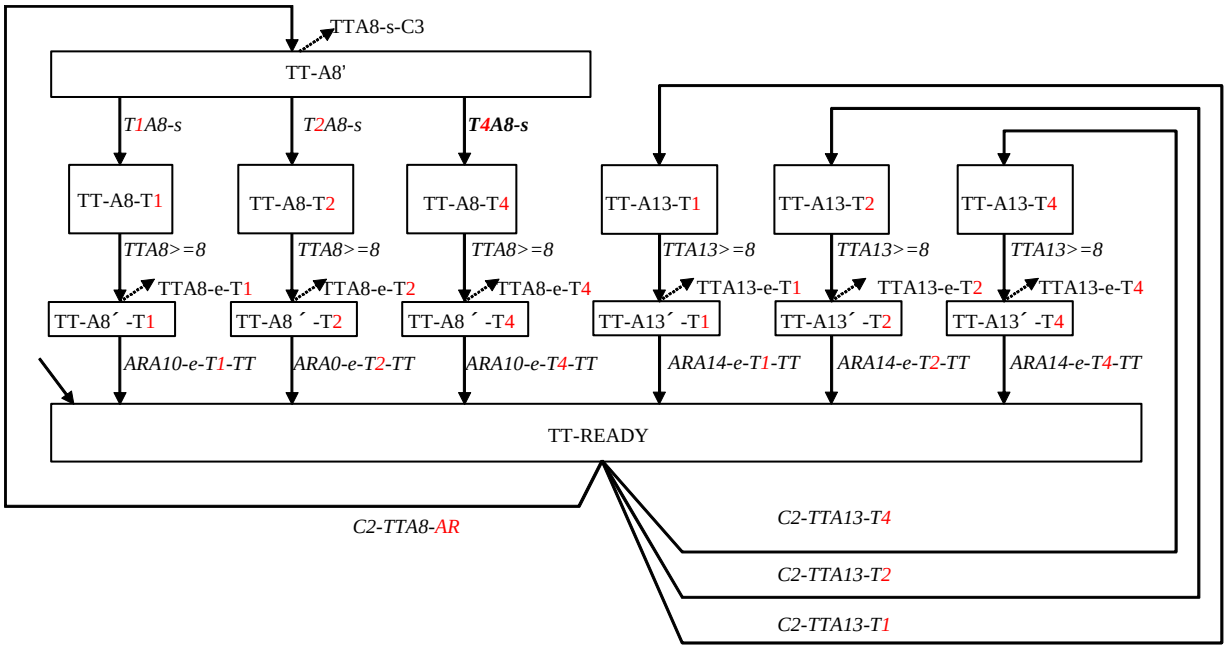
アームロボット（A）



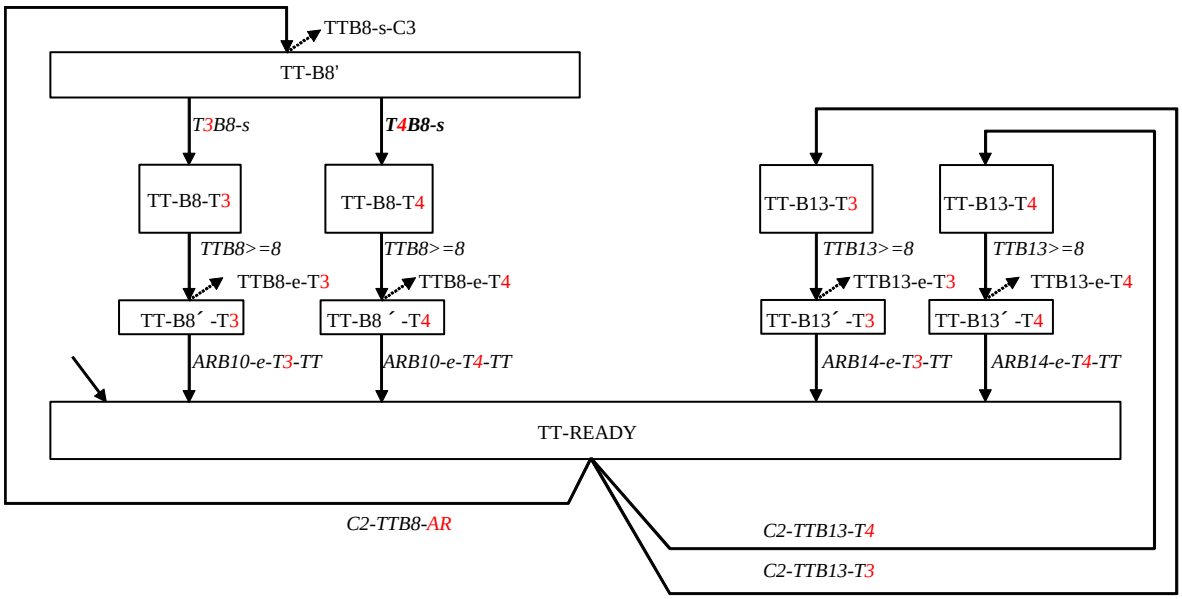
アームロボット（B）



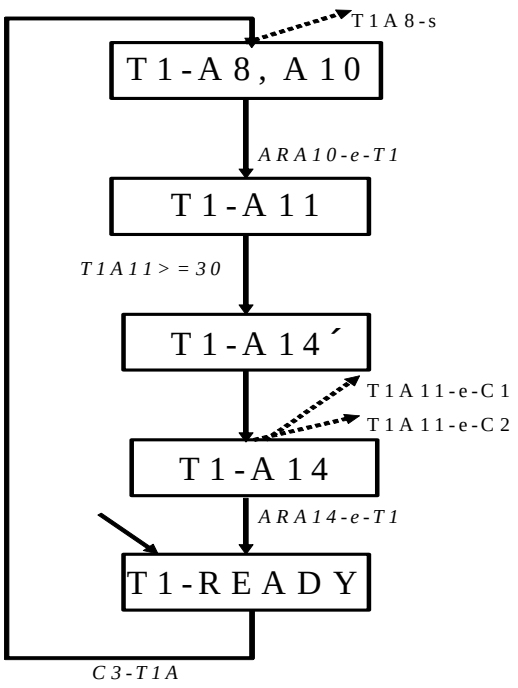
ターンテーブル（A）



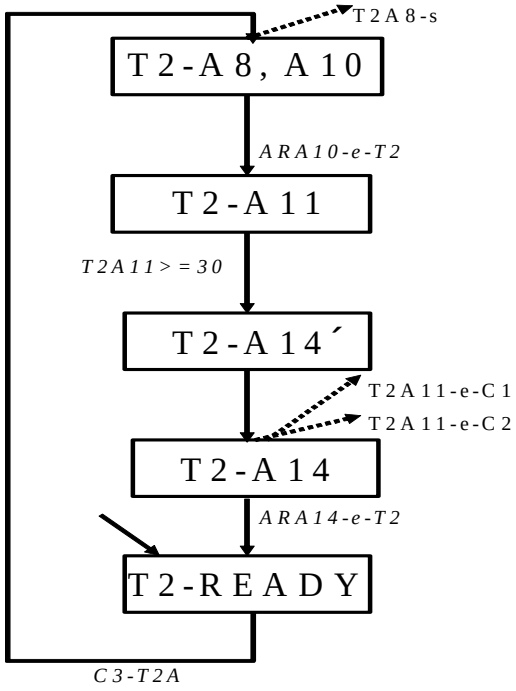
ターンテーブル（B）



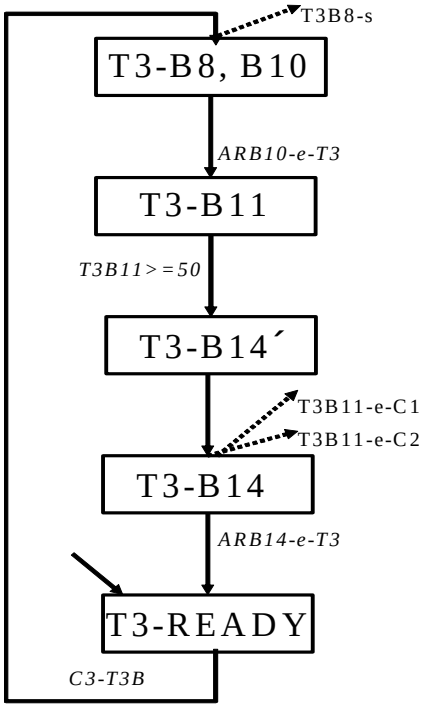
装置Ｔ１



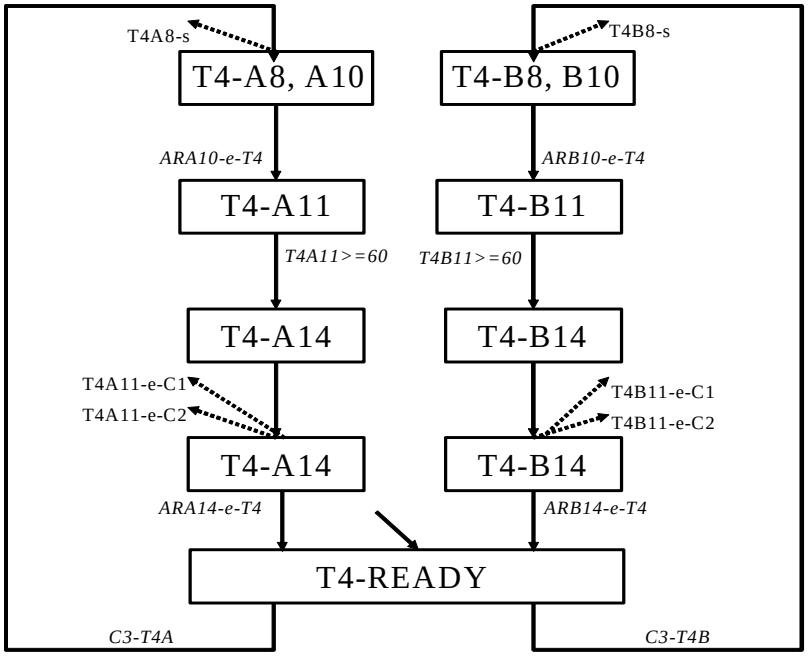
装置Ｔ２



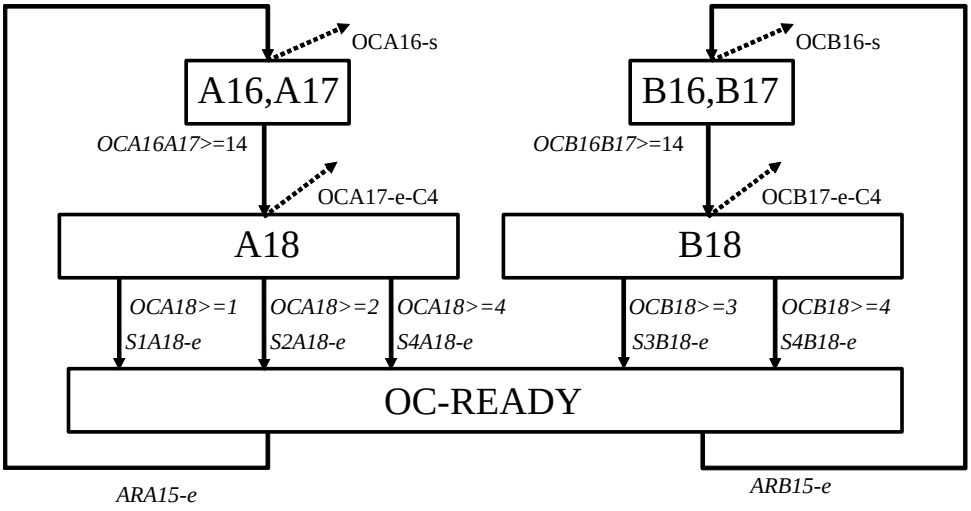
装置Ｔ３



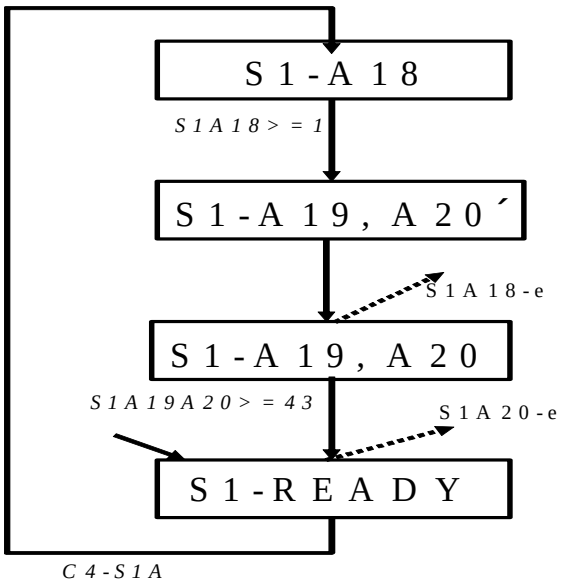
装置Ｔ４



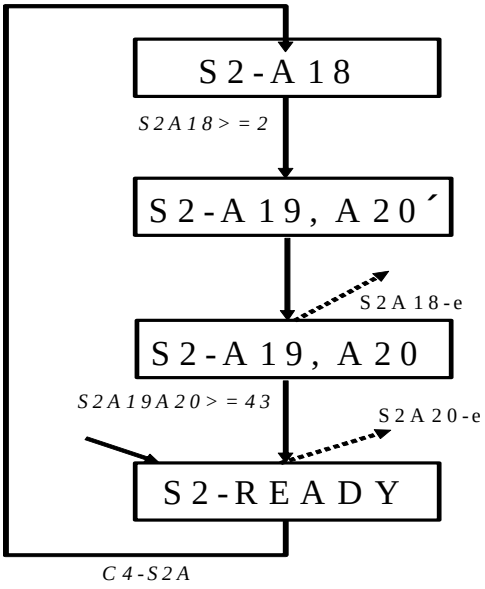
搬出コンベア



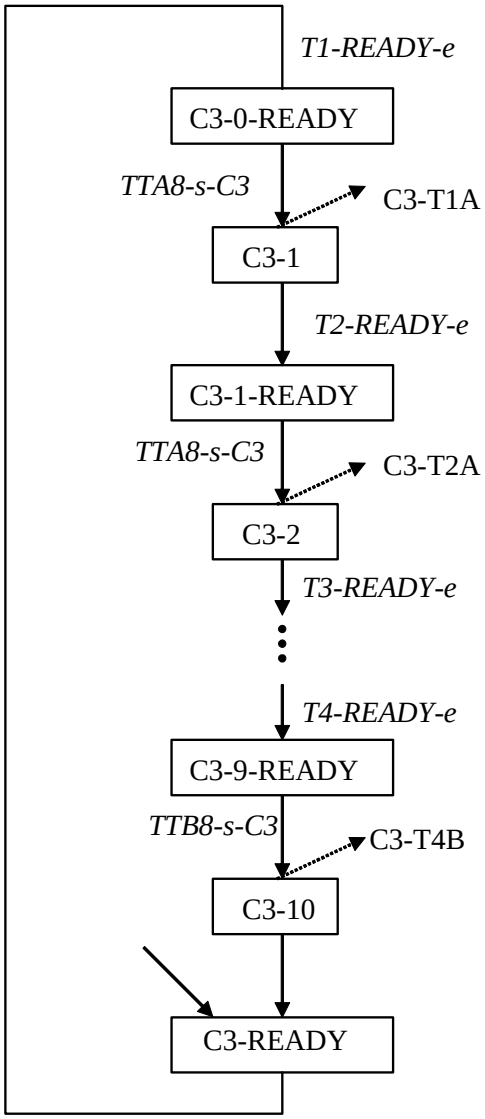
搬出コンベア＆ストッカー１



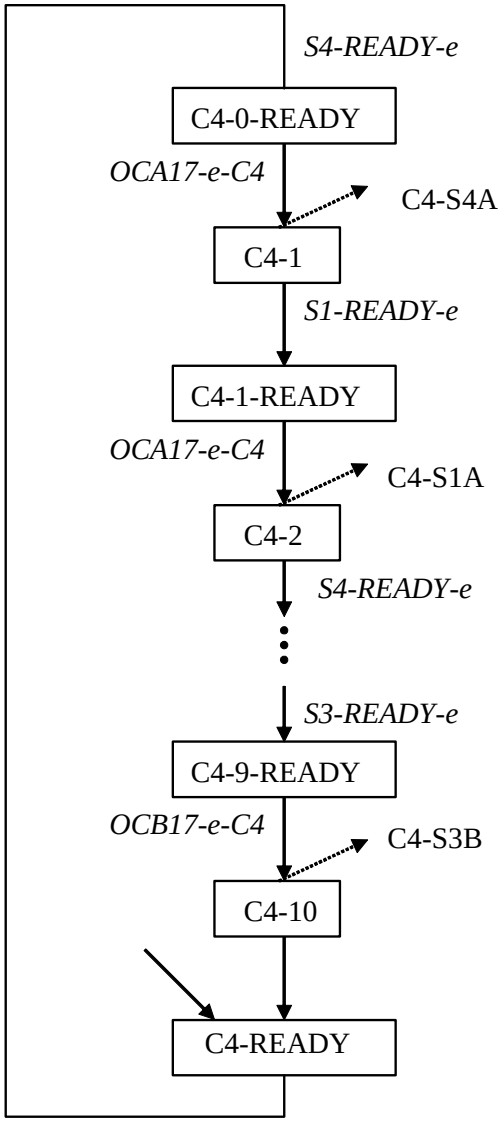
搬入コンベア＆ストッカー２



制御モジュール３



制御モジュール４



A1.4　モジュール抽出システムへの入力データ (ETSC)

•Transfering

1. 遷移関係の識別子　2. 遷移元ロケーション名

3. 遷移先ロケーション名　4. トリガイベント名*

5. 当該遷移実行時に実機へ渡すべき特定の「命令コマンド系列」を指す識別子*

6. 生成イベント名 * 対応するものがない場合は　null

```
T-PFA1  PF-READY        PF-A1   PDAT-e  A1
T-PFA2  PF-A1   PF-A1-2 null    null    PFA1-e 1
T-PFA3  PF-A1-2         PF-A2   PPA2-s  null
T-PFA0  PF-A2   PF-READY        PPA2-e  null
T-PFB1  PF-READY        PF-B1   PDBT-e  A1
T-PFB2  PF-B1   PF-B1-2 null    null    PFB1-e 1
T-PFB3  PF-B1-2         PF-B2   PPB2-s  null
T-PFB0  PF-B2   PF-READY        PPB2-e  null
T-PPA1  PP-READY        PP-A2   PFA1-e  A2      PPA2-s 1
T-PPA2  PP-A2   PP-A3   null    A3      PPA2-e 1
T-PPA3  PP-A3   PP-A4-2 null    null    PPA4-s 1
T-PPA4  PP-A4-2         PP-A4   ICA4-s  A4
T-PPA0  PP-A4   PP-READY        null    null
T-PPB1  PP-READY        PP-B2   PFB1-e  A2      PPB2-s 1
T-PPB2  PP-B2   PP-B3   null    A3      PPB2-e 1
T-PPB3  PP-B3   PP-B4-2 null    null    PPB4-s 1
T-PPB4  PP-B4-2         PP-B4   ICB4-s  A4
T-PPB0  PP-B4   PP-READY        null    null
T-ICA1  IC-READY        IC-A4A5 PPA4-s  null    ICA4-s  1
T-ICA2  IC-A4A5         IC-A7   IC-endLine      null    ICA5-e-C1       1
T-ICA0  IC-A7   IC-READY        ARA7-e          null
T-ICB1  IC-READY        IC-B4B5 PPB4-s  null    ICB4-s  1
T-ICB2  IC-B4B5 IC-B7   IC-endLine      null    ICB5-e-C1       1
T-ICB0  IC-B7   IC-READY        ARB7-e  null
T-ARA1-1AR-READY        AR-A6   C1-ARA6-IC      A5
T-ARA1-2AR-A6   AR-A7   null    A6      ARA6-e-C2       1
T-ARA1-3AR-A7   AR-A9   null    A7      ARA7-e  1
T-ARA1-4-1      AR-A9   AR-A10-T1       TTA8-e-T1       A8
T-ARA1-4-2      AR-A9   AR-A10-T2       TTA8-e-T2       A8
T-ARA1-4-4      AR-A9   AR-A10-T4       TTA8-e-T4       A8
T-ARA1-0-1      AR-A10-T1       AR-READY        null    null    ARA10-e-T1-TT   1
ARA10-e-T1      1
T-ARA1-0-2      AR-A10-T2       AR-READY        null    null    ARA10-e-T2-TT   1
ARA10-e-T2      1
T-ARA1-0-4      AR-A10-T4       AR-READY        null    null    ARA10-e-T4-TT   1
ARA10-e-T4      1
T-ARA2-1-1      AR-READY        AR-A12  C1-ARA12-T1     A7
T-ARA2-1-2      AR-READY        AR-A12  C1-ARA12-T2     A7
T-ARA2-1-4      AR-READY        AR-A12  C1-ARA12-T4     A7
T-ARA2-2-1      AR-A12          AR-A14-T1       TTA13-e-T1      A9
T-ARA2-3-1      AR-A14-T1       AR-A15  null    A10     ARA14-e-T1-TT   1       ARA14-e-T1
1
T-ARA2-2-2      AR-A12  AR-A14-T2       TTA13-e-T2      A9
T-ARA2-3-2      AR-A14-T2       AR-A15  null    A10     ARA14-e-T2-TT   1       ARA14-e-T2
1
```

T-ARA2-2-4 AR-A12 AR-A14-T4 TTA13-e-T4 A9
T-ARA2-3-4 AR-A14-T4 AR-A15 null A10 ARA14-e-T4-TT 1 ARA14-e-T4
1
T-ARA2-4 AR-A15 AR-A15-2 null null ARA15-e 1
T-ARA2-5 AR-A15-2 AR-A16 OCA16-s A11
T-ARA2-0 AR-A16 AR-READY null null
T-ARB1-1 AR-READY AR-B6 C1-ARB6-IC A5
T-ARB1-2 AR-B6 AR-B7 null A6 ARB6-e-C2 1
T-ARB1-3 AR-B7 AR-B9 null A7 ARB7-e 1
T-ARB1-4-3 AR-B9 AR-B10-T3 TTB8-e-T3 A8
T-ARB1-0-3 AR-B10-T3 AR-READY null null ARB10-e-T3-TT 1
ARB10-e-T3 1
T-ARB1-4-1 AR-B9 AR-B10-T4 TTB8-e-T4 A8
T-ARB1-0-4 AR-B10-T4 AR-READY null null ARB10-e-T4-TT 1
ARB10-e-T4 1
T-ARB2-1-3 AR-READY AR-B12 C1-ARB12-T3 A7
T-ARB2-1-4 AR-READY AR-B12 C1-ARB12-T4 A7
T-ARB2-2-3 AR-B12 AR-B14-T3 TTB13-e-T3 A9
T-ARB2-3-3 AR-B14-T3 AR-B15 null A10 ARB14-e-T3-TT 1 ARB14-e-T3
1
T-ARB2-2-4 AR-B12 AR-B14-T4 TTB13-e-T4 A9
T-ARB2-3-4 AR-B14-T4 AR-B15 null A10 ARB14-e-T4-TT 1 ARB14-e-T4
1
T-ARB2-4 AR-B15 AR-B15-2 null null ARB15-e 1
T-ARB2-5 AR-B15-2 AR-B16 OCB16-s A11
T-ARB2-0 AR-B16 AR-READY null null
T-TTA1-1-1 TT-READY TT-A8-2 C2-TTA8-AR null TTA8-s-C3 1
T-TTA1-2-1 TT-A8-2 TT-A8-T1 T1A8-s A12
T-TTA1-3-1 TT-A8-T1 TT-A8-2-T1 null null TTA8-e-T1 1
T-TTA1-0-1 TT-A8-2-T1 TT-READY ARA10-e-T1-TT null
T-TTA1-2-2 TT-A8-2 TT-A8-T2 T2A8-s A13
T-TTA1-3-2 TT-A8-T2 TT-A8-2-T2 null null TTA8-e-T2 1
T-TTA1-0-2 TT-A8-2-T2 TT-READY ARA10-e-T2-TT null
T-TTA1-2-4 TT-A8-2 TT-A8-T4 T4A8-s A14
T-TTA1-3-4 TT-A8-T4 TT-A8-2-T4 null null TTA8-e-T4 1
T-TTA1-0-4 TT-A8-2-T4 TT-READY ARA10-e-T4-TT null
T-TTA2-1-1 TT-READY TT-A13-T1 C2-TTA13-T1 A12
T-TTA2-2-1 TT-A13-T1 TT-A13-2-T1 null null TTA13-e-T1 1
T-TTA2-0-1 TT-A13-2-T1 TT-READY ARA14-e-T1-TT null
T-TTA2-1-2 TT-READY TT-A13-T2 C2-TTA13-T2 A13
T-TTA2-2-2 TT-A13-T2 TT-A13-2-T2 null null TTA13-e-T2 1
T-TTA2-0-2 TT-A13-2-T2 TT-READY ARA14-e-T2-TT null
T-TTA2-1-4 TT-READY TT-A13-T4 C2-TTA13-T4 A14
T-TTA2-2-4 TT-A13-T4 TT-A13-2-T4 null null TTA13-e-T4 1
T-TTA2-0-4 TT-A13-2-T4 TT-READY ARA14-e-T4-TT null
T-TTB1-1-1 TT-READY TT-B8-2 C2-TTB8-AR null TTB8-s-C3 1
T-TTB1-2-3 TT-B8-2 TT-B8-T3 T3B8-s A15
T-TTB1-3-3 TT-B8-T3 TT-B8-2-T3 null null TTB8-e-T3 1
T-TTB1-0-3 TT-B8-2-T3 TT-READY ARB10-e-T3-TT null
T-TTB1-1-4 TT-B8-2 TT-B8-T4 T4B8-s A14
T-TTB1-2-4 TT-B8-T4 TT-B8-2-T4 null null TTB8-e-T4 1
T-TTB1-0-4 TT-B8-2-T4 TT-READY ARB10-e-T4-TT null
T-TTB2-1-3 TT-READY TT-B13-T3 C2-TTB13-T3 A15
T-TTB2-2-3 TT-B13-T3 TT-B13-2-T3 null null TTB13-e-T3 1
T-TTB2-0-3 TT-B13-2-T3 TT-READY ARB14-e-T3-TT null
T-TTB2-1-4 TT-READY TT-B13-T4 C2-TTB13-T4 A14
T-TTB2-2-4 TT-B13-T4 TT-B13-2-T4 null null TTB13-e-T4 1
T-TTB2-0-4 TT-B13-2-T4 TT-READY ARB14-e-T4-TT null
T-T1A1 T1-READY T1-A8A10 C3-T1A null T1A8-s 1
T-T1A2 T1-A8A10 T1-A11-2 ARA10-e-T1 A16
T-T1A2-2 T1-A11-2 T1-A11 T1ATimer-e null

T-T1A3 T1-A11 T1-A14 null null T1A11-e-C1 1 T1A11-e-C2 1
T-T1A0 T1-A14 T1-READY ARA14-e-T1 null
T-T2A1 T2-READY T2-A8A10 C3-T2A null T2A8-s 1
T-T2A2 T2-A8A10 T2-A11-2 ARA10-e-T2 A17
T-T2A2-2 T2-A11-2 T2-A11 T2ATimer-e null
T-T2A3 T2-A11 T2-A14 null null T2A11-e-C1 1 T2A11-e-C2 1
T-T2A0 T2-A14 T2-READY ARA14-e-T2 null
T-T3B1 T3-READY T3-B8B10 C3-T3B null T3B8-s 1
T-T3B2 T3-B8B10 T3-B11-2 ARB10-e-T3 A18
T-T3B2-2 T3-B11-2 T3-B11 T3BTimer-e null
T-T3B3 T3-B11 T3-B14 null null T3B11-e-C1 1 T3B11-e-C2 1
T-T3B0 T3-B14 T3-READY ARB14-e-T3 null
T-T4A1 T4-READY T4-A8A10 C3-T4A null T4A8-s 1
T-T4A2 T4-A8A10 T4-A11-2 ARA10-e-T4 A19
T-T4A2-2 T4-A11-2 T4-A11 T4ATimer-e null
T-T4A3 T4-A11 T4-A14 null null T4A11-e-C1 1 T4A11-e-C2 1
T-T4A0 T4-A14 T4-READY ARA14-e-T4 null
T-T4B1 T4-READY T4-B8B10 C3-T4B null T4B8-s 1
T-T4B2 T4-B8B10 T4-B11-2 ARB10-e-T4 A20
T-T4B2-2 T4-B11-2 T4-B11 T4BTimer-e null
T-T4B3 T4-B11 T4-B14 null null T4B11-e-C1 1 T4B11-e-C2 1
T-T4B0 T4-B14 T4-READY ARB14-e-T4 null
T-OCA1 OC-READY OC-A16A17 ARA15-e A21 OCA16-s 1
T-OCA2 OC-A16A17 OC-A18 null null OCA17-e-C4 1
T-OCA0-S1 OC-A18 OC-READY S1A18-e null
T-OCA0-S2 OC-A18 OC-READY S2A18-e null
T-OCA0-S4 OC-A18 OC-READY S4A18-e null
T-OCB1 OC-READY OC-B16B17 ARB15-e A21 OCB16-s 1
T-OCB2 OC-B16B17 OC-B18 null null OCB17-e-C4 1
T-OCB0-S3 OC-B18 OC-READY S3B18-e null
T-OCB0-S4 OC-B18 OC-READY S4B18-e null
T-S1A1 S1-READY S1-A18 C4-S1A A22
T-S1A2 S1-A18 S1-A19A20-2 null A23 S1A18-e 1
T-S1A2-2 S1-A19A20-2 S1-A19A20 S1ATimer-e null
T-S1A0 S1-A19A20 S1-READY null null S1A20-e 1
T-S2A1 S2-READY S2-A18 C4-S2A A24
T-S2A2 S2-A18 S2-A19A20-2 null A25 S2A18-e 1
T-S2A2-2 S2-A19A20-2 S2-A19A20 S2ATimer-e null
T-S2A0 S2-A19A20 S2-READY null null S2A20-e 1
T-S3B1 S3-READY S3-B18 C4-S3B A26
T-S3B2 S3-B18 S3-B19B20-2 null A27 S3B18-e 1
T-S3B2-2 S3-B19B20-2 S3-B19B20 S3BTimer-e null
T-S3B0 S3-B19B20 S3-READY null null S3B20-e 1

・CommandList

1. 識別子

2. 他のモジュールから指定され実機に渡される命令コマンド系列 （以下“命令コマンド系列”）

A1 Call PartsFeeder goEnd
A2 Call PickPlace getWork
A3 Call PickPlace back
A4 Interfere conv1-Interference Call PickPlace putAndBack Non-Interfere conv1-Interference
A5 Call AirRobot turnLeftPosition
A6 Call AirRobot getWork Non-Interfere conv1-Interference
A7 Call AirRobot turnCenterPosition

A8 Call AirRobot putWork
A9 Call AirRobot getWork
A10 Call AirRobot turnRightPosition
A11 Interfere conv2-Interference Call AirRobot putWork Non-Interfere conv2-Interference
A12 Call TurnTable turnAtDevice1
A13 Call TurnTable turnAtDevice2
A14 Call TurnTable turnAtDevice4
A15 Call TurnTable turnAtDevice3
A16 Reset T1ATimer
A17 Reset T2ATimer
A18 Reset T3BTimer
A19 Reset T4ATimer
A20 Reset T4BTimer
A21 Call Conveyer2 go
A22 Call StockercarryWork1
A23 Reset S1ATimer
A24 Call StockercarryWork2
A25 Reset S2ATimer
A26 Call StockercarryWork3
A27 Reset S3BTimer
A28 Call StockercarryWork4
A29 Reset S4ATimer
A30 Reset S4BTimer

・Interfere

1. 割り込み名（識別子） 2. 割り込み開始/終了

3. 割り込み開始/終了時に実行されるべき命令コマンド系列

conv1-Interference start Call Conveyer1 stop
conv1-Interference end Call Conveyer1 advance
conv2-Interference start Call Conveyer2 stop
conv2-Interference end Call Conveyer2 advance
conv3-Interference start Call Conveyer3 stop
conv3-Interference end Call Conveyer3 advance

・Timer

1. タイマー名 2. カウント時間 3. タイムアウト時に実行されるべき命令コマンド系列

T1ATimer30000 Broad T1ATimer-e 1
T2ATimer30000 Broad T2ATimer-e 1
T3BTimer50000 Broad T3BTimer-e 1
T4ATimer60000 Broad T4ATimer-e 1
T4BTimer60000 Broad T4BTimer-e 1
S1ATimer40000 Call Stocker dropWork1 Broad S1ATimer-e 1
S2ATimer40000 Call Stocker dropWork2 Broad S2ATimer-e 1

```
S3BTimer30000    Call    Stocker  dropWork3        Broad    S3BTimer-e 1
S4ATimer50000    Call    Stocker  dropWork4        Broad    S4ATimer-e 1
S4BTimer50000    Call    Stocker  dropWork4        Broad    S4BTimer-e 1
```

・WatchEvent

1. 当該イベントを監視すべき状況 （どのロケーションがアクティブであるときに監視すべきか)

2. 当該イベントを出力するデバイス名 （どのデバイスからの出力イベントを監視すべきか)

3. 監視イベント名　　4. 当該イベントの発生確認時に実行されるべき命令コマンドリスト

```
IC-READY Conveyer1 watchEndLine   Interfere conv1-Interference      Broad    IC-endLine 1
OC-special  Conveyer2 watchEndLine        Interfere conv2-Interference      Broad    OC-endLine 2
```

・ControlList

1. 当該装置が使用可能となったタイミングで特定の主体から出されるイベント

2. 次の使用許可を特定の主体へ与えるイベント （どのデバイスへ許可を与えるか？）

3. 未実行→1, 実行済み→0

<初期データの例> （青字部分は再スケジューリング後差し替えられる部分）

```
//アームロボット使用順序
ICB5-e-C1 C1-ARB6-IC 1
ICB5-e-C1 C1-ARB6-IC 1
ICA5-e-C1 C1-ARA6-IC 1
ICA5-e-C1 C1-ARA6-IC 1
T4B11-e-C1 C1-ARB12-T4 1
T3B11-e-C1 C1-ARB12-T3 1
ICA5-e-C1 C1-ARA6-IC 1
T2A11-e-C1 C1-ARA12-T2 1
T1A11-e-C1 C1-ARA12-T1 1
ICA5-e-C1 C1-ARA6-IC 1
T4A11-e-C1 C1-ARA12-T4 1
T2A11-e-C1 C1-ARA12-T2 1

//ターンテーブル位置設定順序
ARB6-e-C2 C2-TTB8-AR 1
ARB6-e-C2 C2-TTB8-AR 1
ARA6-e-C2 C2-TTA8-AR 1
ARA6-e-C2 C2-TTA8-AR 1
T4B11-e-C2 C2-TTB13-T4 1
T3B11-e-C2 C2-TTB13-T3 1
ARA6-e-C2 C2-TTA8-AR 1
T2A11-e-C2 C2-TTA13-T2 1
T1A11-e-C2 C2-TTA13-T1 1
ARA6-e-C2 C2-TTA8-AR 1
T4A11-e-C2 C2-TTA13-T4 1
```

```
T2A11-e-C2 C2-TTA13-T2 1

//処理装置割当て
TTB8-s-C3 C3-T4B 1
TTB8-s-C3 C3-T3B 1
TTA8-s-C3 C3-T2A 1
TTA8-s-C3 C3-T1A 1
TTA8-s-C3 C3-T4A 1
TTA8-s-C3 C3-T2A 1

//ストッカー割り当て
OCB17-e-C4 C4-S3B 1
OCB17-e-C4 C4-S4B 1
OCA17-e-C4 C4-S2A 1
OCA17-e-C4 C4-S1A 1
OCA17-e-C4 C4-S4A 1
OCA17-e-C4 C4-S2A 1
```

〈再スケジューリング後の差し替えデータの例〉

```
//アームロボット使用順序
T4B11-e-C1 C1-ARB12-T4 1
T2A11-e-C1 C1-ARA12-T2 1
T1A11-e-C1 C1-ARA12-T1 1
T3B11-e-C1 C1-ARB12-T3 1
ICA5-e-C1 C1-ARA6-IC 1
ICA5-e-C1 C1-ARA6-IC 1
T4A11-e-C1 C1-ARA12-T4 1
T2A11-e-C1 C1-ARA12-T2 1

//ターンテーブル位置設定順序
T4B11-e-C2 C2-TTB13-T4 1
T2A11-e-C2 C2-TTA13-T2 1
T1A11-e-C2 C2-TTA13-T1 1
T3B11-e-C2 C2-TTB13-T3 1
ARA6-e-C2 C2-TTA8-AR 1
ARA6-e-C2 C2-TTA8-AR 1
T4A11-e-C2 C2-TTA13-T4 1
T2A11-e-C2 C2-TTA13-T2 1

//処理装置割当て
TTA8-s-C3 C3-T4A 1
TTA8-s-C3 C3-T2A 1

//ストッカー割り当て
OCB17-e-C4 C4-S4B 1
OCA17-e-C4 C4-S2A 1
OCA17-e-C4 C4-S1A 1
OCB17-e-C4 C4-S3B 1
OCA17-e-C4 C4-S4A 1
OCA17-e-C4 C4-S2A 1
```

A1.5　モジュール抽出システムからの出力データ（実行モジュール）

・Process module

```
// ○プロセス（パーツフィーダ A1 {PF-READY→PF-A1}）
<Process>
        <Precondition> PF-READY </Precondition>
        <PreOutputEvent> PDAT-e </PreOutputEvent>
        <Command> Leave PF-READY </Command>
        <Command> Call PartsFeeder goEnd </Command>
        <Command> Enter PF-A1 </Command>
</Process>

// ○プロセス（パーツフィーダ A2 {PF-A1→PF-A1'}）
<Process>
        <Precondition> PF-A1 </Precondition>
        <Command> Leave PF-A1 </Command>
        <Command> Broad PFA1-e 1 </Command>
        <Command> Enter PF-A1' </Command>
</Process>

// ○プロセス（パーツフィーダ 3 {PF-A1'→PF-A2}）
<Process>
        <Precondition> PF-A1' </Precondition>
        <PreOutputEvent> PPA2-s </PreOutputEvent>
        <Command> Leave PF-A1' </Command>
        <Command> Enter PF-A2 </Command>
</Process>

// ○プロセス（パーツフィーダ 0 {PF-A2→PF-READY}）
<Process>
        <Precondition> PF-A2 </Precondition>
        <PreOutputEvent> PPA2-e </PreOutputEvent>
        <Command> Leave PF-A2 </Command>
        <Command> Enter PF-READY </Command>
</Process>

// ○プロセス（パーツフィーダ B1  {PF-READY→PF-B1)
<Process>
        <Precondition> PF-READY </Precondition>
        <PreOutputEvent> PDBT-e </PreOutputEvent>
        <Command> Leave PF-READY </Command>
        <Command> Call PartsFeeder goEnd </Command>
        <Command> Enter PF-B1 m</Command>
</Process>

// ○プロセス（パーツフィーダ B2  {PF-B1→PF-B1'}）
<Process>
        <Precondition> PF-B1 </Precondition>
        <Command> Leave PF-B1 </Command>
        <Command> Broad PFB1-e 1 </Command>
```

```
        <Command> Enter PF-B1’ </Command>
</Process>

// ○プロセス（パーツフィーダ B3 {PF-B1’→PF-B2}）
<Process>
        <Precondition> PF-B1’ </Precondition>
        <PreOutputEvent> PPB2-s </PreOutputEvent>
        <Command> Leave PF-B1’ </Command>
        <Command> Enter PF-B2 </Command>
</Process>

// ○プロセス（パーツフィーダ B0 {PF-B2→PF-READY}）
<Process>
        <Precondition> PF-B2 </Precondition>
        <PreOutputEvent> PPB2-e </PreOutputEvent>
        <Command> Leave PF-B2 </Command>
        <Command> Enter PF-READY </Command>
</Process>

// ○プロセス（ピック＆プレース A1 {PP-READY→PP-A2}）
<Process>
        <Precondition> PP-READY </Precondition>
        <PreOutputEvent> PFA1-e </PreOutputEvent>
        <Command> Leave PP-READY </Command>
        <Command> Broad PPA2-s 1 </Command>
        <Command> Call PickPlace getWork </Command>
        <Command> Enter PP-A2 </Command>
</Process>

// ○プロセス（ピック＆プレース A2 {PP-A2→PP-A3}）
<Process>
        <Precondition> PP-A2 </Precondition>
        <Command> Leave PP-A2 </Command>
        <Command> Broad PPA2-e 1 </Command>
        <Command> Call PickPlace back </Command>
        <Command> Enter PP-A3 </Command>
</Process>

// ○プロセス（ピック＆プレース A3 {PP-A3→PP-A4’}）
<Process>
        <Precondition> PP-A3 </Precondition>
        <Command> Leave PP-A3 </Command>
        <Command> Broad PPA4-s 1 </Command>
        <Command> Enter PP-A4’ </Command>
</Process>

// ○プロセス（ピック＆プレース A3 {PP-A4’→PP-A4}）
<Process>
        <Precondition> PP-A4’ </Precondition>
        <PreOutputEvent> ICA4-s </PreOutputEvent>
        <Command> Leave PP-A4’ </Command>
        <Command> Interfere conv1-Interference </Command>
        <Command> Call PickPlace putAndBack </Command>
```

```
        <Command> Non-Interfere conv1-Interference </Command>
        <Command> Enter PP-A4 </Command>
</Process>

// ○プロセス（ピック＆プレース A0 {PP-A4→PP-READY}）
<Process>
        <Precondition> PP-A4 </Precondition>
        <Command> Leave PP-A4 </Command>
        <Command> Enter PP-READY </Command>
</Process>

// ○プロセス（ピック＆プレース B1 {PP-READY→PP-B2}）
<Process>
        <Precondition> PP-READY </Precondition>
        <PreOutputEvent> PFB1-e </PreOutputEvent>
        <Command> Leave PP-READY </Command>
        <Command> Broad PPB2-s 1 </Command>
        <Command> Call PickPlace getWork </Command>
        <Command> Enter PP-B2 </Command>
</Process>

// ○プロセス（ピック＆プレース B2 {PP-B2→PP-B3}）
<Process>
        <Precondition> PP-B2 </Precondition>
        <Command> Leave PP-B2 </Command>
        <Command> Broad PPB2-e 1 </Command>
        <Command> Call PickPlace back </Command>
        <Command> Enter PP-B3 </Command>
</Process>

// ○プロセス（ピック＆プレース B3 {PP-B3→PP-B4'}）
<Process>
        <Precondition> PP-B3 </Precondition>
        <Command> Leave PP-B3 </Command>
        <Command> Broad PPB4-s 1 </Command>
        <Command> Enter PP-B4' </Command>
</Process>

// ○プロセス（ピック＆プレース B3 {PP-B4'→PP-B4}）
<Process>
        <Precondition> PP-B4' </Precondition>
        <PreOutputEvent> ICB4-s </PreOutputEvent>
        <Command> Leave PP-B4' </Command>
        <Command> Interfere conv1-Interference </Command>
        <Command> Call PickPlace putAndBack </Command>
        <Command> Non-Interfere conv1-Interference </Command>
        <Command> Enter PP-B4 </Command>
</Process>

// ○プロセス（ピック＆プレース B0 {PP-B4→PP-READY}）
<Process>
        <Precondition> PP-B4 </Precondition>
        <Command> Leave PP-B4 </Command>
```

```
        <Command> Enter PP-READY </Command>
</Process>

// ○プロセス（搬入コンベア A1 {IC-READY→IC-A4,A5}）
<Process>
        <Precondition> IC-READY </Precondition>
        <PreOutputEvent> PPA4-s </PreOutputEvent>
        <Command> Leave IC-READY </Command>
        <Command> Broad ICA4-s 1 </Command>
        <Command> Enter IC-A4A5 </Command>
</Process>

// ○プロセス（搬入コンベア A2 {IC-A4,A5→IC-A7}）
<Process>
        <Precondition> IC-A4A5 </Precondition>
        <PreOutputEvent> IC-endLine </PreOutputEvent>
        <Command> Leave IC-A4A5 </Command>
        <Command> Broad ICA5-e-C1 1 </Command>
        <Command> Enter IC-A7 </Command>
</Process>

// ○プロセス（搬入コンベア A0 {IC-A7→IC-READY}）
<Process>
        <Precondition> IC-A7 </Precondition>
        <PreOutputEvent> ARA7-e </PreOutputEvent>
        <Command> Leave IC-A7 </Command>
        <Command> Enter IC-READY </Command>
</Process>

// ○プロセス（搬入コンベア B1 {IC-READY→IC-B4,B5}）
<Process>
        <Precondition> IC-READY </Precondition>
        <PreOutputEvent> PPB4-s </PreOutputEvent>
        <Command> Leave IC-READY </Command>
        <Command> Broad ICB4-s 1 </Command>
        <Command> Enter IC-B4B5 </Command>
</Process>

// ○プロセス（搬入コンベア B2 {IC-B4,B5→IC-B7}）
<Process>
        <Precondition> IC-B4B5 </Precondition>
        <PreOutputEvent> IC-endLine </PreOutputEvent>
        <Command> Leave IC-B4B5 </Command>
        <Command> Broad ICB5-e-C1 1 </Command>
        <Command> Enter IC-B7 </Command>
</Process>

// ○プロセス（搬入コンベア B0 {IC-B7→IC-READY}）
<Process>
        <Precondition> IC-B7 </Precondition>
        <PreOutputEvent> ARB7-e </PreOutputEvent>
        <Command> Leave IC-B7 </Command>
        <Command> Enter IC-READY </Command>
```

```
</Process>

// ○プロセス（エアロボ A1-1 {AR-READY→AR-A6}）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> C1-ARA6-IC </PreOutputEvent>
        <Command> Leave AR-READY </Command>
        <Command> Call AirRobot turnLeftPosition </Command>
        <Command> Enter AR-A6 </Command>
</Process>

// ○プロセス（エアロボ A1-2 {AR-A6→AR-A7}）
<Process>
        <Precondition> AR-A6 </Precondition>
        <Command> Leave AR-A6 </Command>
        <Command> Broad ARA6-e-C2 1 </Command>
        <Command> Call AirRobot getWork </Command>
        <Command> Non-Interfere conv1-Interference </Command>
        <Command> Enter AR-A7 </Command>
</Process>

// ○プロセス（エアロボ A1-3 {AR-A7→AR-A9}）
<Process>
        <Precondition> AR-A7 </Precondition>
        <Command> Leave AR-A7 </Command>
        <Command> Broad ARA7-e 1 </Command>
        <Command> Call AirRobot turnCenterPosition </Command>
        <Command> Enter AR-A9 </Command>
</Process>

// ○プロセス（エアロボ A1-4-1 {AR-A9→AR-A10-T1}）
<Process>
        <Precondition> AR-A9 </Precondition>
        <PreOutputEvent> TTA8-e-T1 </PreOutputEvent>
        <Command> Leave AR-A9 </Command>
        <Command> Call AirRobot putWork </Command>
        <Command> Enter AR-A10-T1 </Command>
</Process>

// ○プロセス（エアロボ A1-4-2 {AR-A9→AR-A10-T2}）
<Process>
        <Precondition> AR-A9 </Precondition>
        <PreOutputEvent> TTA8-e-T2 </PreOutputEvent>
        <Command> Leave AR-A9 </Command>
        <Command> Call AirRobot putWork </Command>
        <Command> Enter AR-A10-T2 </Command>
</Process>

// ○プロセス（エアロボ A1-4-4 {AR-A9→AR-A10-T4}）
<Process>
        <Precondition> AR-A9 </Precondition>
        <PreOutputEvent> TTA8-e-T4 </PreOutputEvent>
        <Command> Leave AR-A9 </Command>
```

```
        <Command> Call AirRobot putWork </Command>
        <Command> Enter AR-A10-T4 </Command>
</Process>

// ○プロセス（エアロボ A1-0-1 {AR-A10-T1→AR-READY}）
<Process>
        <Precondition> AR-A10-T1 </Precondition>
        <Command> Leave AR-A10-T1 </Command>
        <Command> Broad ARA10-e-T1 1 </Command>
        <Command> Broad ARA10-e-T1-TT 1 </Command>
        <Command> Enter AR-READY </Command>
</Process>

// ○プロセス（エアロボ A1-0-1 {AR-A10-T2→AR-READY}）
<Process>
        <Precondition> AR-A10-T2 </Precondition>
        <Command> Leave AR-A10-T2 </Command>
        <Command> Broad ARA10-e-T2 1 </Command>
        <Command> Broad ARA10-e-T2-TT 1 </Command>
        <Command> Enter AR-READY </Command>
</Process>

// ○プロセス（エアロボ A1-0-1 {AR-A10-T4→AR-READY}）
<Process>
        <Precondition> AR-A10-T4 </Precondition>
        <Command> Leave AR-A10-T4 </Command>
        <Command> Broad ARA10-e-T4 1 </Command>
        <Command> Broad ARA10-e-T4-TT 1 </Command>
        <Command> Enter AR-READY </Command>
</Process>

// ○プロセス（エアロボ A2-1-T1 {AR-READY→AR-A12}）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> C1-ARA12-T1 </PreOutputEvent>
        <Command> Leave AR-READY </Command>
        <Command> Call AirRobot turnCenterPosition </Command>
        <Command> Enter AR-A12 </Command>
</Process>

// ○プロセス（エアロボ A2-2-T1 {AR-A12→AR-A14-T1}）
<Process>
        <Precondition> AR-A12 </Precondition>
        <PreOutputEvent> TTA13-e-T1 </PreOutputEvent>
        <Command> Leave AR-A12 </Command>
        <Command> Call AirRobot getWork </Command>
        <Command> Enter AR-A14-T1 </Command>
</Process>

// ○プロセス（エアロボ A2-3-T1 {AR-A14-T1→AR-A15}）
<Process>
        <Precondition> AR-A14-T1 </Precondition>
        <Command> Leave AR-A14-T1 </Command>
```

```
        <Command> Broad ARA14-e-T1 1 </Command>
        <Command> Broad ARA14-e-T1-TT 1 </Command>
        <Command> Call AirRobot turnRightPosition </Command>
        <Command> Enter AR-A15 </Command>
</Process>

// ○プロセス（エアロボ A2-1-T2 {AR-READY→AR-A12}）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> C1-ARA12-T2 </PreOutputEvent>
        <Command> Leave AR-READY </Command>
        <Command> Call AirRobot turnCenterPosition </Command>
        <Command> Enter AR-A12 </Command>
</Process>

// ○プロセス（エアロボ A2-2-T2 {AR-A12→AR-A14-T2}）
<Process>
        <Precondition> AR-A12 </Precondition>
        <PreOutputEvent> TTA13-e-T2 </PreOutputEvent>
        <Command> Leave AR-A12 </Command>
        <Command> Call AirRobot getWork </Command>
        <Command> Enter AR-A14-T2 </Command>
</Process>

// ○プロセス（エアロボ A2-3-T2 {AR-A14-T2→AR-A15}）
<Process>
        <Precondition> AR-A14-T2 </Precondition>
        <Command> Leave AR-A14-T2 </Command>
        <Command> Broad ARA14-e-T2 1 </Command>
        <Command> Broad ARA14-e-T2-TT 1 </Command>
        <Command> Call AirRobot turnRightPosition </Command>
        <Command> Enter AR-A15
 </Command>
</Process>

// ○プロセス（エアロボ A2-1-T4 {AR-READY→AR-A12}）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> C1-ARA12-T4 </PreOutputEvent>
        <Command> Leave AR-READY </Command>
        <Command> Call AirRobot turnCenterPosition </Command>
        <Command> Enter AR-A12 </Command>
</Process>

// ○プロセス（エアロボ A2-2-T4 {AR-A12→AR-A14-T4}）
<Process>
        <Precondition> AR-A12 </Precondition>
        <PreOutputEvent> TTA13-e-T4 </PreOutputEvent>
        <Command> Leave AR-A12 </Command>
        <Command> Call AirRobot getWork </Command>
        <Command> Enter AR-A14-T4 </Command>
</Process>

// ○プロセス（エアロボ A2-3-T4 {AR-A14-T4→AR-A15}）
<Process>
        <Precondition> AR-A14-T4 </Precondition>
```

```
        <Command> Leave AR-A14-T4 </Command>
        <Command> Broad ARA14-e-T4 1 </Command>
        <Command> Broad ARA14-e-T4-TT 1 </Command>
        <Command> Call AirRobot turnRightPosition </Command>
        <Command> Enter AR-A15 </Command>
</Process>

// ○プロセス（エアロボ A2-4 {AR-A15→AR-A15'}）
<Process>
        <Precondition> AR-A15 </Precondition>
        <Command> Leave AR-A15 </Command>
        <Command> Broad ARA15-e 1 </Command>
        <Command> Enter AR-A15' </Command>
</Process>

// ○プロセス（エアロボ A2-5 {AR-A15'→AR-A16}）
<Process>
        <Precondition> AR-A15' </Precondition>
        <PreOutputEvent> OCA16-s </PreOutputEvent>
        <Command> Leave AR-A15' </Command>
        <Command> Interfere conv2-Interference </Command>
        <Command> Call AirRobot putWork </Command>
        <Command> Non-Interfere conv2-Interference </Command>
        <Command> Enter AR-A16 </Command>
</Process>

// ○プロセス（エアロボ A2-0 {AR-A16→AR-READY}）
<Process>
        <Precondition> AR-A16 </Precondition>
        <Command> Leave AR-A16 </Command>
        <Command> Enter AR-READY </Command>
</Process>

// ○プロセス（エアロボ B1-1 {AR-READY→AR-B6}）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> C1-ARB6-IC </PreOutputEvent>
        <Command> Leave AR-READY </Command>
        <Command> Call AirRobot turnLeftPosition </Command>
        <Command> Enter AR-B6 </Command>
</Process>

// ○プロセス（エアロボ B1-2 {AR-B6→AR-B7}）
<Process>
        <Precondition> AR-B6 </Precondition>
        <Command> Leave AR-B6 </Command>
        <Command> Broad ARB6-e-C2 1 </Command>
        <Command> Call AirRobot getWork </Command>
        <Command> Non-Interfere conv1-Interference </Command>
        <Command> Enter AR-B7 </Command>
</Process>

// ○プロセス（エアロボ B1-3 {AR-B7→AR-B9}）
<Process>
        <Precondition> AR-B7 </Precondition>
        <Command> Leave AR-B7 </Command>
```

```
        <Command> Broad ARB7-e 1 </Command>
        <Command> Call AirRobot turnCenterPosition </Command>
        <Command> Enter AR-B9 </Command>
</Process>

// ○プロセス（エアロボ B1-4-3 {AR-B9→AR-B10-T3}）
<Process>
        <Precondition> AR-B9 </Precondition>
        <PreOutputEvent> TTB8-e-T3 </PreOutputEvent>
        <Command> Leave AR-B9 </Command>
        <Command> Call AirRobot putWork </Command>
        <Command> Enter AR-B10-T3 </Command>
</Process>

// ○プロセス（エアロボ B1-0-3 {AR-B10-T3→AR-READY}）
<Process>
        <Precondition> AR-B10-T3 </Precondition>
        <Command> Leave AR-B10-T3 </Command>
        <Command> Broad ARB10-e-T3 1 </Command>
        <Command> Broad ARB10-e-T3-TT 1 </Command>
        <Command> Enter AR-READY </Command>
</Process>

// ○プロセス（エアロボ B1-4-1 {AR-B9→AR-B10-T4}）
<Process>
        <Precondition> AR-B9 </Precondition>
        <PreOutputEvent> TTB8-e-T4 </PreOutputEvent>
        <Command> Leave AR-B9 </Command>
        <Command> Call AirRobot putWork </Command>
        <Command> Enter AR-B10-T4 </Command>
</Process>

// ○プロセス（エアロボ B1-0-4 {AR-B10-T1→AR-READY}）
<Process>
        <Precondition> AR-B10-T4 </Precondition>
        <Command> Leave AR-B10-T4 </Command>
        <Command> Broad ARB10-e-T4 1 </Command>
        <Command> Broad ARB10-e-T4-TT 1 </Command>
        <Command> Enter AR-READY </Command>
</Process>

// ○プロセス（エアロボ B2-1-T3 {AR-READY→AR-B12}）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> C1-ARB12-T3 </PreOutputEvent>
        <Command> Leave AR-READY </Command>
        <Command> Call AirRobot turnCenterPosition </Command>
        <Command> Enter AR-B12 </Command>
</Process>

// ○プロセス（エアロボ B2-2-T3 {AR-B12→AR-B14-T3}）
<Process>
        <Precondition> AR-B12 </Precondition>
        <PreOutputEvent> TTB13-e-T3 </PreOutputEvent>
        <Command> Leave AR-B12 </Command>
        <Command> Call AirRobot    getWork </Command>
```

```
        <Command> Enter AR-B14-T3 </Command>
</Process>

// ○プロセス（エアロボ B2-3-T3 {AR-B14-T3→AR-B15}）
<Process>
        <Precondition> AR-B14-T3 </Precondition>
        <Command> Leave AR-B14-T3 </Command>
        <Command> Broad ARB14-e-T3 1 </Command>
        <Command> Broad ARB14-e-T3-TT 1 </Command>
        <Command> Call AirRobot turnRightPosition </Command>
        <Command> Enter AR-B15 </Command>
</Process>

/ ○プロセス（エアロボ B2-1-T4 {AR-READY→AR-B12}）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> C1-ARB12-T4 </PreOutputEvent>
        <Command> Leave AR-READY </Command>
        <Command> Call AirRobot turnCenterPosition </Command>
        <Command> Enter AR-B12 </Command>
</Process>

// ○プロセス（エアロボ B2-2-T4 {AR-B12→AR-B14-T4}）
<Process>
        <Precondition> AR-B12 </Precondition>
        <PreOutputEvent> TTB13-e-T4 </PreOutputEvent>
        <Command> Leave AR-B12 </Command>
        <Command> Call AirRobot    getWork </Command>
        <Command> Enter AR-B14-T4 </Command>
</Process>

// ○プロセス（エアロボ B2-3-T4 {AR-B14-T4→AR-B15}）
<Process>
        <Precondition> AR-B14-T4 </Precondition>
        <Command> Leave AR-B14-T4 </Command>
        <Command> Broad ARB14-e-T4 1 </Command>
        <Command> Broad ARB14-e-T4-TT 1 </Command>
        <Command> Call AirRobot turnRightPosition </Command>
        <Command> Enter AR-B15 </Command>
</Process>

// ○プロセス（エアロボ B2-4 {AR-B15→AR-B15'}）
<Process>
        <Precondition> AR-B15 </Precondition>
        <Command> Leave AR-B15 </Command>
        <Command> Broad ARB15-e 1 </Command>
        <Command> Enter AR-B15' </Command>
</Process>

// ○プロセス（エアロボ B2-5 {AR-B15'→AR-B16}）
<Process>
        <Precondition> AR-B15' </Precondition>
        <PreOutputEvent> OCB16-s </PreOutputEvent>
        <Command> Leave AR-B15' </Command>
        <Command> Interfere conv2-Interference </Command>
        <Command> Call AirRobot putWork </Command>
```

```
        <Command> Non-Interfere conv2-Interference </Command>
        <Command> Enter AR-B16 </Command>
</Process>

// ○プロセス（エアロボ B2-0 {AR-B16→AR-READY}）
<Process>
        <Precondition> AR-B16 </Precondition>
        <Command> Leave AR-B16 </Command>
        <Command> Enter AR-READY </Command>
</Process>

// ○プロセス（ターンテーブル A1-1-T1 {TT-READY→TT-A8'}）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> C2-TTA8-AR </PreOutputEvent>
        <Command> Leave TT-READY </Command>
        <Command> Broad TTA8-s-C3 1 </Command>
        <Command> Enter TT-A8' </Command>
</Process>

// ○プロセス（ターンテーブル A1-2-T1 {TT-A8'→TT-A8-T1}）
<Process>
        <Precondition> TT-A8' </Precondition>
        <PreOutputEvent> T1A8-s </PreOutputEvent>
        <Command> Leave TT-A8' </Command>
        <Command> Call TurnTable turnAtDevice1 </Command>
        <Command> Enter TT-A8-T1 </Command>
</Process>

// ○プロセス（ターンテーブル A1-3-T1 {TT-A8-T1→TT-A8'-T1}）
<Process>
        <Precondition> TT-A8-T1 </Precondition>
        <Command> Leave TT-A8-T1 </Command>
        <Command> Broad TTA8-e-T1 1 </Command>
        <Command> Enter TT-A8'-T1 </Command>
</Process>

// ○プロセス（ターンテーブル A1-0-T1 {TT-A8'-T1→TT-READY}）
<Process>
        <Precondition> TT-A8'-T1 </Precondition>
        <PreOutputEvent> ARA10-e-T1-TT </PreOutputEvent>
        <Command> Leave TT-A8'-T1 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル A1-2-T2 {TT-A8'→TT-A8-T2}）
<Process>
        <Precondition> TT-A8' </Precondition>
        <PreOutputEvent> T2A8-s </PreOutputEvent>
        <Command> Leave TT-A8' </Command>
        <Command> Call TurnTable turnAtDevice2 </Command>
        <Command> Enter TT-A8-T2 </Command>
</Process>

// ○プロセス（ターンテーブル A1-3-T2 {TT-A8-T2→TT-A8'-T2}）
<Process>
```

```
        <Precondition> TT-A8-T2 </Precondition>
        <Command> Leave TT-A8-T2 </Command>
        <Command> Broad TTA8-e-T2 1 </Command>
        <Command> Enter TT-A8'-T2 </Command>
</Process>

// ○プロセス（ターンテーブル A1-0-T2 {TT-A8'-T2→TT-READY}）
<Process>
        <Precondition> TT-A8'-T2 </Precondition>
        <PreOutputEvent> ARA10-e-T2-TT </PreOutputEvent>
        <Command> Leave TT-A8'-T2 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル A1-2-T4 {TT-A8'→TT-A8-T4}）
<Process>
        <Precondition> TT-A8' </Precondition>
        <PreOutputEvent> T4A8-s </PreOutputEvent>
        <Command> Leave TT-A8' </Command>
        <Command> Call TurnTable turnAtDevice4 </Command>
        <Command> Enter TT-A8-T4 </Command>
</Process>

// ○プロセス（ターンテーブル A1-3-T4 {TT-A8-T4→TT-A8'-T4}）
<Process>
        <Precondition> TT-A8-T4 </Precondition>
        <Command> Leave TT-A8-T4 </Command>
        <Command> Broad TTA8-e-T4 1 </Command>
        <Command> Enter TT-A8'-T4 </Command>
</Process>

// ○プロセス（ターンテーブル A1-0-T4 {TT-A8'-T4→TT-READY}）
<Process>
        <Precondition> TT-A8'-T4 </Precondition>
        <PreOutputEvent> ARA10-e-T4-TT </PreOutputEvent>
        <Command> Leave TT-A8'-T4 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル A2-1-T1 {TT-READY→TT-A13-T1}）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> C2-TTA13-T1 </PreOutputEvent>
        <Command> Leave TT-READY </Command>
        <Command> Call TurnTable turnAtDevice1 </Command>
        <Command> Enter TT-A13-T1 </Command>
</Process>

// ○プロセス（ターンテーブル A2-2-T1 {TT-A13-T1→TT-A13'-T1}）
<Process>
        <Precondition> TT-A13-T1 </Precondition>
        <Command> Leave TT-A13-T1 </Command>
        <Command> Broad TTA13-e-T1 1 </Command>
        <Command> Enter TT-A13'-T1 </Command>
</Process>
```

```
// ○プロセス（ターンテーブル A2-0-T1 {TT-A13'-T1→TT-READY}）
<Process>
        <Precondition> TT-A13'-T1 </Precondition>
        <PreOutputEvent> ARA14-e-T1-TT </PreOutputEvent>
        <Command> Leave TT-A13'-T1 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル A2-1-T2 {TT-READY→TT-A13-T2}）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> C2-TTA13-T2 </PreOutputEvent>
        <Command> Leave TT-READY </Command>
        <Command> Call TurnTable turnAtDevice2 </Command>
        <Command> Enter TT-A13-T2 </Command>
</Process>

// ○プロセス（ターンテーブル A2-2-T2 {TT-A13-T2→TT-A13'-T2}）
<Process>
        <Precondition> TT-A13-T2 </Precondition>
        <Command> Leave TT-A13-T2 </Command>
        <Command> Broad TTA13-e-T2 1 </Command>
        <Command> Enter TT-A13'-T2 </Command>
</Process>

// ○プロセス（ターンテーブル A2-0-T2 {TT-A13'-T2→TT-READY}）
<Process>
        <Precondition> TT-A13'-T2 </Precondition>
        <PreOutputEvent> ARA14-e-T2-TT </PreOutputEvent>
        <Command> Leave TT-A13'-T2 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル A2-1-T4 {TT-READY→TT-A13-T4}）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> C2-TTA13-T4 </PreOutputEvent>
        <Command> Leave TT-READY </Command>
        <Command> Call TurnTable turnAtDevice4 </Command>
        <Command> Enter TT-A13-T4 </Command>
</Process>

// ○プロセス（ターンテーブル A2-1-T4 {TT-A13-T4→TT-A13'-T4}）
<Process>
        <Precondition> TT-A13-T4 </Precondition>
        <Command> Leave TT-A13-T4 </Command>
        <Command> Broad TTA13-e-T4 1 </Command>
        <Command> Enter TT-A13'-T4 </Command>
</Process>

// ○プロセス（ターンテーブル A2-0-T4 {TT-A13-T4→TT-READY}）
<Process>
        <Precondition> TT-A13'-T4 </Precondition>
        <PreOutputEvent> ARA14-e-T4-TT </PreOutputEvent>
        <Command> Leave TT-A13'-T4 </Command>
        <Command> Enter TT-READY </Command>
```

```
</Process>

// ○プロセス（ターンテーブル B1-1-T1 {TT-READY→TT-B8'}）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> C2-TTB8-AR </PreOutputEvent>
        <Command> Leave TT-READY </Command>
        <Command> Broad TTB8-s-C3 1 </Command>
        <Command> Enter TT-B8' </Command>
</Process>

// ○プロセス（ターンテーブル B1-2-T3 {TT-B8' →TT-B8-T3}）
<Process>
        <Precondition> TT-B8' </Precondition>
        <PreOutputEvent> T3B8-s </PreOutputEvent>
        <Command> Leave TT-B8' </Command>
        <Command> Call TurnTable turnAtDevice3 </Command>
        <Command> Enter TT-B8-T3 </Command>
</Process>

// ○プロセス（ターンテーブル B1-3-T3 {TT-B8-T3→TT-B8'-T3}）
<Process>
        <Precondition> TT-B8-T3 </Precondition>
        <Command> Leave TT-B8-T3 </Command>
        <Command> Broad TTB8-e-T3 1 </Command>
        <Command> Enter TT-B8'-T3 </Command>
</Process>

// ○プロセス（ターンテーブル B1-0-T3 {TT-B8'-T3→TT-READY}）
<Process>
        <Precondition> TT-B8'-T3 </Precondition>
        <PreOutputEvent> ARB10-e-T3-TT </PreOutputEvent>
        <Command> Leave TT-B8'-T3 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル B1-1-T4 {TT-B8' →TT-B8-T4}）
<Process>
        <Precondition> TT-B8' </Precondition>
        <PreOutputEvent> T4B8-s </PreOutputEvent>
        <Command> Leave TT-B8' </Command>
        <Command> Call TurnTable turnAtDevice4 </Command>
        <Command> Enter TT-B8-T4 </Command>
</Process>

// ○プロセス（ターンテーブル B1-2-T4 {TT-B8-T4→TT-B8'-T4}）
<Process>
        <Precondition> TT-B8-T4 </Precondition>
        <Command> Leave TT-B8-T4 </Command>
        <Command> Broad TTB8-e-T4 1 </Command>
        <Command> Enter TT-B8'-T4 </Command>
</Process>

// ○プロセス（ターンテーブル B1-0-T4 {TT-B8'-T4→TT-READY}）
<Process>
        <Precondition> TT-B8'-T4 </Precondition>
```

```
        <PreOutputEvent> ARB10-e-T4-TT </PreOutputEvent>
        <Command> Leave TT-B8'-T4 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル B2-1-T3 {TT-READY→TT-B13-T3}）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> C2-TTB13-T3 </PreOutputEvent>
        <Command> Leave TT-READY </Command>
        <Command> Call TurnTable turnAtDevice3 </Command>
        <Command> Enter TT-B13-T3 </Command>
</Process>

// ○プロセス（ターンテーブル B2-2-T3 {TT-B13-T3→TT-B13'-T3}）
<Process>
        <Precondition> TT-B13-T3 </Precondition>
        <Command> Leave TT-B13-T3 </Command>
        <Command> Broad TTB13-e-T3 1 </Command>
        <Command> Enter TT-B13'-T3 </Command>
</Process>

// ○プロセス（ターンテーブル B2-0-T3 {TT-B13'-T3→TT-READY}）
<Process>
        <Precondition> TT-B13'-T3 </Precondition>
        <PreOutputEvent> ARB14-e-T3-TT </PreOutputEvent>
        <Command> Leave TT-B13'-T3 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（ターンテーブル B2-1-T4 {TT-READY→TT-B13-T1}）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> C2-TTB13-T4 </PreOutputEvent>
        <Command> Leave TT-READY </Command>
        <Command> Call TurnTable turnAtDevice4 </Command>
        <Command> Enter TT-B13-T4 </Command>
</Process>

// ○プロセス（ターンテーブル B2-2-T4 {TT-B13-T4→TT-B13'-T4}）
<Process>
        <Precondition> TT-B13-T4 </Precondition>
        <Command> Leave TT-B13-T4 </Command>
        <Command> Broad TTB13-e-T4 1 </Command>
        <Command> Enter TT-B13'-T4 </Command>
</Process>

// ○プロセス（ターンテーブル B2-0-T4 {TT-B13-T4→TT-READY}）
<Process>
        <Precondition> TT-B13'-T4 </Precondition>
        <PreOutputEvent> ARB14-e-T4-TT </PreOutputEvent>
        <Command> Leave TT-B13'-T4 </Command>
        <Command> Enter TT-READY </Command>
</Process>

// ○プロセス（装置 T1A1 {T1-READY→T1-A8,A10}）
```

```
<Process>
        <Precondition> T1-READY </Precondition>
        <PreOutputEvent> C3-T1A </PreOutputEvent>
        <Command> Leave T1-READY </Command>
        <Command> Broad T1A8-s 1 </Command>
        <Command> Enter T1-A8A10 </Command>
</Process>

// ○プロセス（装置 T1A2 {T1-A8,A10→(T1-A11)}）
<Process>
        <Precondition> T1-A8A10 </Precondition>
        <PreOutputEvent> ARA10-e-T1 </PreOutputEvent>
        <Command> Leave T1-A8A10 </Command>
        <Command> Reset T1ATimer </Command>
</Process>

// ○プロセス（装置 T1A3 {T1-A11→T1-A14}）
<Process>
        <Precondition> T1-A11 </Precondition>
        <Command> Leave T1-A11 </Command>
        <Command> Broad T1A11-e-C1 1 </Command>
        <Command> Broad T1A11-e-C2 1 </Command>
        <Command> Enter T1-A14 </Command>
</Process>

// ○プロセス（装置 T1A0 {T1-A14→T1-READY}）
<Process>
        <Precondition> T1-A14 </Precondition>
        <PreOutputEvent> ARA14-e-T1 </PreOutputEvent>
        <Command> Leave T1-A14 </Command>
        <Command> Enter T1-READY </Command>
</Process>

// ○プロセス（装置 T2A1 {T2-READY→T2-A8,A10}）
<Process>
        <Precondition> T2-READY </Precondition>
        <PreOutputEvent> C3-T2A </PreOutputEvent>
        <Command> Leave T2-READY </Command>
        <Command> Broad T2A8-s 1 </Command>
        <Command> Enter T2-A8A10 </Command>
</Process>

// ○プロセス（装置 T2A2 {T2-A8,A10→(T2-A11)}）
<Process>
        <Precondition> T2-A8A10 </Precondition>
        <PreOutputEvent> ARA10-e-T2 </PreOutputEvent>
        <Command> Leave T2-A8A10 </Command>
        <Command> Reset T2ATimer </Command>
</Process>

// ○プロセス（装置 T2A3 {T2-A11→T2-A14}）
<Process>
        <Precondition> T2-A11 </Precondition>
        <Command> Leave T2-A11 </Command>
        <Command> Broad T2A11-e-C1 1 </Command>
        <Command> Broad T2A11-e-C2 1 </Command>
```

```
        <Command> Enter T2-A14 </Command>
</Process>

// ○プロセス（装置 T2A0 {T2-A14→T2-READY}）
<Process>
        <Precondition> T2-A14 </Precondition>
        <PreOutputEvent> ARA14-e-T2 </PreOutputEvent>
        <Command> Leave T2-A14 </Command>
        <Command> Enter T2-READY </Command>
</Process>

// ○プロセス（装置 T3B1 {T3-READY→T3-B8,B10}）
<Process>
        <Precondition> T3-READY </Precondition>
        <PreOutputEvent> C3-T3B </PreOutputEvent>
        <Command> Leave T3-READY </Command>
        <Command> Broad T3B8-s 1 </Command>
        <Command> Enter T3-B8B10 </Command>
</Process>

// ○プロセス（装置 T3B2 {T3-B8,B10→(T3-B11)}）
<Process>
        <Precondition> T3-B8B10 </Precondition>
        <PreOutputEvent> ARB10-e-T3 </PreOutputEvent>
        <Command> Leave T3-B8B10 </Command>
        <Command> Reset T3BTimer </Command>
</Process>

// ○プロセス（装置 T3B0 {T3-B11→T3-B14}）
<Process>
        <Precondition> T3-B11 </Precondition>
        <Command> Leave T3-B11 </Command>
        <Command> Broad T3B11-e-C1 1 </Command>
        <Command> Broad T3B11-e-C2 1 </Command>
        <Command> Enter T3-B14 </Command>
</Process>

// ○プロセス（装置 T3B0 {T3-B14→T3-READY}）
<Process>
        <Precondition> T3-B14 </Precondition>
        <PreOutputEvent> ARB14-e-T3 </PreOutputEvent>
        <Command> Leave T3-B14 </Command>
        <Command> Enter T3-READY </Command>
</Process>

// ○プロセス（装置 T4A1 {T4-READY→T4-A8,A10}）
<Process>
        <Precondition> T4-READY </Precondition>
        <PreOutputEvent> C3-T4A </PreOutputEvent>
        <Command> Leave T4-READY </Command>
        <Command> Broad T4A8-s 1 </Command>
        <Command> Enter T4-A8A10 </Command>
</Process>

// ○プロセス（装置 T4A1 {T4-READY→T4-A8,A10}）
<Process>
```

```
        <Precondition> T4-A8A10 </Precondition>
        <PreOutputEvent> ARA10-e-T4 </PreOutputEvent>
        <Command> Leave T4-A8A10 </Command>
        <Command> Reset T4ATimer </Command>
</Process>

// ○プロセス（装置 T4A3 {T4-A11→T4-A14}）
<Process>
        <Precondition> T4-A11 </Precondition>
        <Command> Leave T4-A11 </Command>
        <Command> Broad T4A11-e-C1 1 </Command>
        <Command> Broad T4A11-e-C2 1 </Command>
        <Command> Enter T4-A14 </Command>
</Process>

// ○プロセス（装置 T4A0 {T4-A14→T4-READY}）
<Process>
        <Precondition> T4-A14 </Precondition>
        <PreOutputEvent> ARA14-e-T4 </PreOutputEvent>
        <Command> Leave T4-A14 </Command>
        <Command> Enter T4-READY </Command>
</Process>

// ○プロセス（装置 T4B1 {T4-READY→T4-B8,B10}）
<Process>
        <Precondition> T4-READY </Precondition>
        <PreOutputEvent> C3-T4B </PreOutputEvent>
        <Command> Leave T4-READY </Command>
        <Command> Broad T4B8-s 1 </Command>
        <Command> Enter T4-B8B10 </Command>
</Process>

// ○プロセス（装置 T4B2 {T4-B8,B10→(T4-B11)}）
<Process>
        <Precondition> T4-B8B10 </Precondition>
        <PreOutputEvent> ARB10-e-T4 </PreOutputEvent>
        <Command> Leave T4-B8B10 </Command>
        <Command> Reset T4BTimer </Command>
</Process>

// ○プロセス（装置 T4B3 {T4-B11→T4-B14}）
<Process>
        <Precondition> T4-B11 </Precondition>
        <Command> Leave T4-B11 </Command>
        <Command> Broad T4B11-e-C1 1 </Command>
        <Command> Broad T4B11-e-C2 1 </Command>
        <Command> Enter T4-B14 </Command>
</Process>

// ○プロセス（装置 T4B0 {T4-B14→T4-READY}）
<Process>
        <Precondition> T4-B14 </Precondition>
        <PreOutputEvent> ARB14-e-T4 </PreOutputEvent>
        <Command> Leave T4-B14 </Command>
        <Command> Enter T4-READY </Command>
</Process>
```

```
// ○プロセス（搬出コンベア A1 {OC-READY→OC-A16,A17}）
<Process>
        <Precondition> OC-READY </Precondition>
        <PreOutputEvent> ARA15-e </PreOutputEvent>
        <Command> Leave OC-READY </Command>
        <Command> Broad OCA16-s 1 </Command>
        <Command> Call Conveyer2 go </Command>
        <Command> Enter OC-A16A17 </Command>
</Process>

// ○プロセス（搬出コンベア A2 {OC-A16,A17→OC-A18}）
<Process>
        <Precondition> OC-A16A17 </Precondition>
        <Command> Leave OC-A16A17 </Command>
        <Command> Broad OCA17-e-C4 1 </Command>
        <Command> Enter OC-A18 </Command>
</Process>

// ○プロセス（搬出コンベア A0-S1 {OC-A18→OC-READY}）
<Process>
        <Precondition> OC-A18 </Precondition>
        <PreOutputEvent> S1A18-e </PreOutputEvent>
        <Command> Leave OC-A18 </Command>
        <Command> Enter OC-READY </Command>
</Process>

// ○プロセス（搬出コンベア A0-S2 {OC-A18→OC-READY}）
<Process>
        <Precondition> OC-A18 </Precondition>
        <PreOutputEvent> S2A18-e </PreOutputEvent>
        <Command> Leave OC-A18 </Command>
        <Command> Enter OC-READY </Command>
</Process>

// ○プロセス（搬出コンベア A0-S4 {OC-A18→OC-READY}）
<Process>
        <Precondition> OC-A18 </Precondition>
        <PreOutputEvent> S4A18-e </PreOutputEvent>
        <Command> Leave OC-A18 </Command>
        <Command> Enter OC-READY </Command>
</Process>

// ○プロセス（搬出コンベア B0 {OC-READY→OC-B16,B17}）
<Process>
        <Precondition> OC-READY </Precondition>
        <PreOutputEvent> ARB15-e </PreOutputEvent>
        <Command> Leave OC-READY </Command>
        <Command> Broad OCB16-s 1 </Command>
        <Command> Call Conveyer2 go </Command>
        <Command> Enter OC-B16B17 </Command>
</Process>

// ○プロセス（搬出コンベア B2 {OC-B16,B17→OC-B18}）
<Process>
        <Precondition> OC-B16B17 </Precondition>
```

```
        <Command> Leave OC-B16B17 </Command>
        <Command> Broad OCB17-e-C4 1 </Command>
        <Command> Enter OC-B18 </Command>
</Process>

// ○プロセス（搬出コンベア B0-S3 {OC-B18→OC-READY}）
<Process>
        <Precondition> OC-B18 </Precondition>
        <PreOutputEvent> S3B18-e </PreOutputEvent>
        <Command> Leave OC-B18 </Command>
        <Command> Enter OC-READY </Command>
</Process>

// ○プロセス（搬出コンベア B0-S4 {OC-B18→OC-READY}）
<Process>
        <Precondition> OC-B18 </Precondition>
        <PreOutputEvent> S4B18-e </PreOutputEvent>
        <Command> Leave OC-B18 </Command>
        <Command> Enter OC-READY </Command>
</Process>

// ○プロセス（ストッカー1A1 {S1-READY→S1-A18}）
<Process>
        <Precondition> S1-READY </Precondition>
        <PreOutputEvent> C4-S1A </PreOutputEvent>
        <Command> Leave S1-READY </Command>
        <Command> Call Stocker carryWork1 </Command>
        <Command> Enter S1-A18 </Command>
</Process>

// ○プロセス（ストッカー1A2 {S1-A18→(S1-A19,A20)}）
<Process>
        <Precondition> S1-A18 </Precondition>
        <Command> Leave S1-A18 </Command>
        <Command> Broad S1A18-e 1 </Command>
        <Command> Reset S1ATimer </Command>
</Process>

// ○プロセス（ストッカー1A3 {S1-A19,A20→S1-READY}）
<Process>
        <Precondition> S1-A19A20 </Precondition>
        <Command> Leave S1-A19A20 </Command>
        <Command> Broad S1A20-e 1 </Command>
        <Command> Enter S1-READY </Command>
</Process>

// ○プロセス（ストッカー2A1 {S2-READY→S2-A18}）
<Process>
        <Precondition> S2-READY </Precondition>
        <PreOutputEvent> C4-S2A </PreOutputEvent>
        <Command> Leave S2-READY </Command>
        <Command> Call Stocker carryWork2 </Command>
        <Command> Enter S2-A18 </Command>
</Process>

// ○プロセス（ストッカー2A2 {S2-A18→(S2-A19,A20)}）
```

```
<Process>
        <Precondition> S2-A18 </Precondition>
        <Command> Leave S2-A18 </Command>
        <Command> Broad S2A18-e 1 </Command>
        <Command> Reset S2ATimer </Command>
</Process>

// ○プロセス（ストッカー2A3 {S2-A19,A20→S2-READY}）
<Process>
        <Precondition> S2-A19A20 </Precondition>
        <Command> Leave S2-A19A20 </Command>
        <Command> Broad S2A20-e 1 </Command>
        <Command> Enter S2-READY </Command>
</Process>

// ○プロセス（ストッカー3-B1 {S3-READY→S3-B18}）
<Process>
        <Precondition> S3-READY </Precondition>
        <PreOutputEvent> C4-S3B </PreOutputEvent>
        <Command> Leave S3-READY </Command>
        <Command> Call Stocker carryWork3 </Command>
        <Command> Enter S3-B18 </Command>
</Process>

// ○プロセス（ストッカー3B2 {S3-B18→(S3-B19,B20)}）
<Process>
        <Precondition> S3-B18 </Precondition>
        <Command> Leave S3-B18 </Command>
        <Command> Broad S3B18-e 1 </Command>
        <Command> Reset S3BTimer </Command>
</Process>

// ○プロセス（ストッカー3B3 {S3-B19,B20→S3-READY}）
<Process>
        <Precondition> S3-B19B20 </Precondition>
        <Command> Leave S3-B19B20 </Command>
        <Command> Broad   S3B20-e 1 </Command>
        <Command> Enter S3-READY </Command>
</Process>

// ○プロセス（ストッカー4A1 {S4-READY→A18}）
<Process>
        <Precondition> S4-READY </Precondition>
        <PreOutputEvent> C4-S4A </PreOutputEvent>
        <Command> Leave S4-READY </Command>
        <Command> Call Stocker carryWork4 </Command>
        <Command> Enter S4-A18 </Command>
</Process>

// ○プロセス（ストッカー4A2 {S4-A18→(S4-A19,A20)}）
<Process>
        <Precondition> S4-A18 </Precondition>
        <Command> Leave S4-A18 </Command>
        <Command> Broad S4A18-e 1 </Command>
        <Command> Reset S4ATimer </Command>
</Process>
```

```
// ○プロセス（ストッカー4A3 {S4-A19,A20→S4-READY}）
<Process>
        <Precondition> S4-A19A20 </Precondition>
        <Command> Leave S4-A19A20 </Command>
        <Command> Broad S4A20-e 1 </Command>
        <Command> Enter S4-READY </Command>
</Process>

// ○プロセス（ストッカー4B1 {S4-READY→S4-B18}）
<Process>
        <Precondition> S4-READY </Precondition>
        <PreOutputEvent> C4-S4B </PreOutputEvent>
        <Command> Leave S4-READY </Command>
        <Command> Call Stocker carryWork4 </Command>
        <Command> Enter S4-B18 </Command>
</Process>

// ○プロセス（ストッカー4B2 {S4-B18→(S4-B19,B20)}）
<Process>
        <Precondition> S4-B18 </Precondition>
        <Command> Leave S4-B18 </Command>
        <Command> Broad S4B18-e 1 </Command>
        <Command> Reset S4BTimer </Command>
</Process>

// ○プロセス（ストッカー4B3 {S4-B19,B20→S4-READY}）
<Process>
        <Precondition> S4-B19B20 </Precondition>
        <Command> Leave S4-B19B20 </Command>
        <Command> Broad S4B20-e 1 </Command>
        <Command> Enter S4-READY </Command>
</Process>

// ○プロセス（制御モジュール 1-1 C1-ARA6-IC）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> ICA5-e-C1 </PreOutputEvent>
        <Command> Refer List1 ICA5-e-C1 </Command>
</Process>

// ○プロセス（制御モジュール 1-2-1 C1-ARA12-T1）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> T1A11-e-C1 </PreOutputEvent>
        <Command> Refer List1 T1A11-e-C1 </Command>
</Process>

// ○プロセス（制御モジュール 1-2-2 C1-ARA12-T2）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> T2A11-e-C1 </PreOutputEvent>
        <Command> Refer List1 T2A11-e-C1 </Command>
</Process>

// ○プロセス（制御モジュール 1-2-4 C1-ARA12-T4）
```

```
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> T4A11-e-C1 </PreOutputEvent>
        <Command> Refer List1 T4A11-e-C1 </Command>
</Process>

// ○プロセス（制御モジュール 1-1 C1-ARB6-IC）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> ICB5-e-C1 </PreOutputEvent>
        <Command> Refer List1 ICB5-e-C1 </Command>
</Process>


// ○プロセス（制御モジュール 1-2-3 C1-ARB12-T3）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> T3B11-e-C1 </PreOutputEvent>
        <Command> Refer List1 T3B11-e-C1 </Command>
</Process>

// ○プロセス（制御モジュール 1-2-4 C1-ARB12-T4）
<Process>
        <Precondition> AR-READY </Precondition>
        <PreOutputEvent> T4B11-e-C1 </PreOutputEvent>
        <Command> Refer List1 T4B11-e-C1 </Command>
</Process>

// ○プロセス（制御モジュール 2-1 C2-TTA8-AR）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> ARA6-e-C2 </PreOutputEvent>
        <Command> Refer List2 ARA6-e-C2 </Command>
</Process>

// ○プロセス（制御モジュール 2-2-1 C2-TTA13-T1）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> T1A11-e-C2 </PreOutputEvent>
        <Command> Refer List2 T1A11-e-C2 </Command>
</Process>

// ○プロセス（制御モジュール 2-2-2 C2-TTA13-T2）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> T2A11-e-C2 </PreOutputEvent>
        <Command> Refer List2 T2A11-e-C2 </Command>
</Process>

// ○プロセス（制御モジュール 2-2-4 C2-TTA13-T4）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> T4A11-e-C2 </PreOutputEvent>
        <Command> Refer List2 T4A11-e-C2 </Command>
</Process>
```

```
// ○プロセス（制御モジュール 2-1 C2-TTB8-AR）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> ARB6-e-C2 </PreOutputEvent>
        <Command> Refer List2 ARB6-e-C2 </Command>
</Process>

// ○プロセス（制御モジュール 2-2-3 C2-TTB13-T3）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> T3B11-e-C2 </PreOutputEvent>
        <Command> Refer List2 T3B11-e-C2 </Command>
</Process>

// ○プロセス（制御モジュール 2-2-4 C2-TTB13-T4）
<Process>
        <Precondition> TT-READY </Precondition>
        <PreOutputEvent> T4B11-e-C2 </PreOutputEvent>
        <Command> Refer List2 T4B11-e-C2 </Command>
</Process>

// ○プロセス（制御モジュール 3-1 C3-T1A）
<Process>
        <Precondition> T1-READY </Precondition>
        <PreOutputEvent> TTA8-s-C3 </PreOutputEvent>
        <Command> Refer List3 TTA8-s-C3 </Command>
</Process>

// ○プロセス（制御モジュール 3-2 C3-T2A）
<Process>
        <Precondition> T2-READY </Precondition>
        <PreOutputEvent> TTA8-s-C3 </PreOutputEvent>
        <Command> Refer List3 TTA8-s-C3 </Command>
</Process>

// ○プロセス（制御モジュール 3-4 C3-T4A）
<Process>
        <Precondition> T4-READY </Precondition>
        <PreOutputEvent> TTA8-s-C3 </PreOutputEvent>
        <Command> Refer List3 TTA8-s-C3 </Command>
</Process>

// ○プロセス（制御モジュール 3-3 C3-T3B）
<Process>
        <Precondition> T3-READY </Precondition>
        <PreOutputEvent> TTB8-s-C3 </PreOutputEvent>
        <Command> Refer List3 TTB8-s-C3 </Command>
</Process>

// ○プロセス（制御モジュール 3-4 C3-T4B）
<Process>
        <Precondition> T4-READY </Precondition>
        <PreOutputEvent> TTB8-s-C3 </PreOutputEvent>
        <Command> Refer List3 TTB8-s-C3 </Command>
</Process>
```

```
// ○プロセス（制御モジュール 4-1 C3-S1A）
<Process>
        <Precondition> S1-READY </Precondition>
        <PreOutputEvent> OCA17-e-C4 </PreOutputEvent>
        <Command> Refer List4 OCA17-e-C4 </Command>
</Process>

// ○プロセス（制御モジュール 4-2 C3-S2A）
<Process>
        <Precondition> S2-READY </Precondition>
        <PreOutputEvent> OCA17-e-C4 </PreOutputEvent>
        <Command> Refer List4 OCA17-e-C4 </Command>
</Process>

// ○プロセス（制御モジュール 4-4 C3-S4A）
<Process>
        <Precondition> S4-READY </Precondition>
        <PreOutputEvent> OCA17-e-C4 </PreOutputEvent>
        <Command> Refer List4 OCA17-e-C4 </Command>
</Process>

// ○プロセス（制御モジュール 4-3 C3-S3B）
<Process>
        <Precondition> S3-READY </Precondition>
        <PreOutputEvent> OCB17-e-C4 </PreOutputEvent>
        <Command> Refer List4 OCB17-e-C4 </Command>
</Process>

// ○プロセス（制御モジュール 4-4 C3-S4B）
<Process>
        <Precondition> S4-READY </Precondition>
        <PreOutputEvent> OCB17-e-C4 </PreOutputEvent>
        <Command> Refer List4 OCB17-e-C4 </Command>
</Process>
```

・Timer module

```
// ○タイマー（T1 での加工時間 A{→T1-A11}）
<Timer>
        <TimerName> T1ATimer </TimerName>
        <CountTime> 30000 </CountTime>
        <Command> Enter T1-A11 </Command>
</Timer>

// ○タイマー（T2 での加工時間 A{→T2-A11}）
<Timer>
        <TimerName> T2ATimer </TimerName>
        <CountTime> 30000 </CountTime>
        <Command> Enter T2-A11 </Command>
</Timer>

// ○タイマー（T3 での加工時間 B{→T3-B11}）
<Timer>
        <TimerName> T3BTimer </TimerName>
        <CountTime> 50000 </CountTime>
```

```
        <Command> Enter T3-B11 </Command>
</Timer>

// ○タイマー（T4 での加工時間 A {→T4-A11}）
<Timer>
        <TimerName> T4ATimer </TimerName>
        <CountTime> 60000 </CountTime>
        <Command> Enter T4-A11 </Command>
</Timer>

// ○タイマー（T4 での加工時間 B {→T4-B11}）
<Timer>
        <TimerName> T4BTimer </TimerName>
        <CountTime> 60000 </CountTime>
        <Command> Enter T4-B11 </Command>
</Timer>

// ○タイマー（S1 での加工時間 A　{→S1-A19,A20}）
<Timer>
        <TimerName> S1ATimer </TimerName>
        <CountTime> 40000 </CountTime>
        <Command> Call Stocker dropWork1 </Command>
        <Command> Enter S1-A19A20 </Command>
</Timer>

// ○タイマー（S2 での加工時間 A {→S2-A19,A20}）
<Timer>
        <TimerName> S2ATimer </TimerName>
        <CountTime> 40000 </CountTime>
        <Command> Call Stocker dropWork2 </Command>
        <Command> Enter S2-A19A20 </Command>
</Timer>

// ○タイマー（S3 での加工時間 B {→S3-B19,B20}）
<Timer>
        <TimerName> S3BTimer </TimerName>
        <CountTime> 30000 </CountTime>
        <Command> Call Stocker dropWork3 </Command>
        <Command> Enter S3-B19B20 </Command>
</Timer>

// ○タイマー（S4 での加工時間 {→S4-A19,A20}）
<Timer>
        <TimerName> S4ATimer </TimerName>
        <CountTime> 50000 </CountTime>
        <Command> Call Stocker dropWork4 </Command>
        <Command> Enter S4-A19A20 </Command>
</Timer>

// ○タイマー（S4 での加工時間 B {→S4-B19,B20}）
<Timer>
        <TimerName> S4BTimer </TimerName>
        <CountTime> 50000 </CountTime>
        <Command> Call Stocker dropWork4 </Command>
        <Command> Enter S4-B19B20 </Command>
</Timer>
```

・Event module

```
// ○イベント（搬入コンベア終点監視）
<Event>
        <EventListener> Conveyer1 watchEndLine </EventListener>
        <Command> Interfere conv1-Interference </Command>
        <Command> Broad IC-endLine 1 </Command>
</Event>
```

・Intervene module

```
// ○割り込み（搬入コンベア）
<Interference>
        <InterferenceName> conv1-Interference </InterferenceName>
        <Command-Before> Call Conveyer1 stop </Command-Before>
        <Command-After> Call Conveyer1 advance </Command-After>
</Interference>

// ○割り込み（搬出コンベア）
<Interference>
        <InterferenceName> conv2-Interference </InterferenceName>
        <Command-Before> Call Conveyer2 stop </Command-Before>
        <Command-After> Call Conveyer2 advance </Command-After>
</Interference>
```

・Control-List module

```
// ○作業数
//<NumOfWark>10</NumOfWark>

// ○ControlList0 (生産要求 AB PDAT-e,PDBT-e)
<ControlList>
<ControlListName> List0 </ControlListName>
<ControlListElement> PDT-e PDAT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDAT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDBT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDBT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDAT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDBT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDBT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDAT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDAT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDBT-e 1 </ControlListElement>
<ControlListElement> PDT-e PDEND-e 1 </ControlListElement>
</ControlList>

// ○ControlList1 (アームロボット使用順序)
```

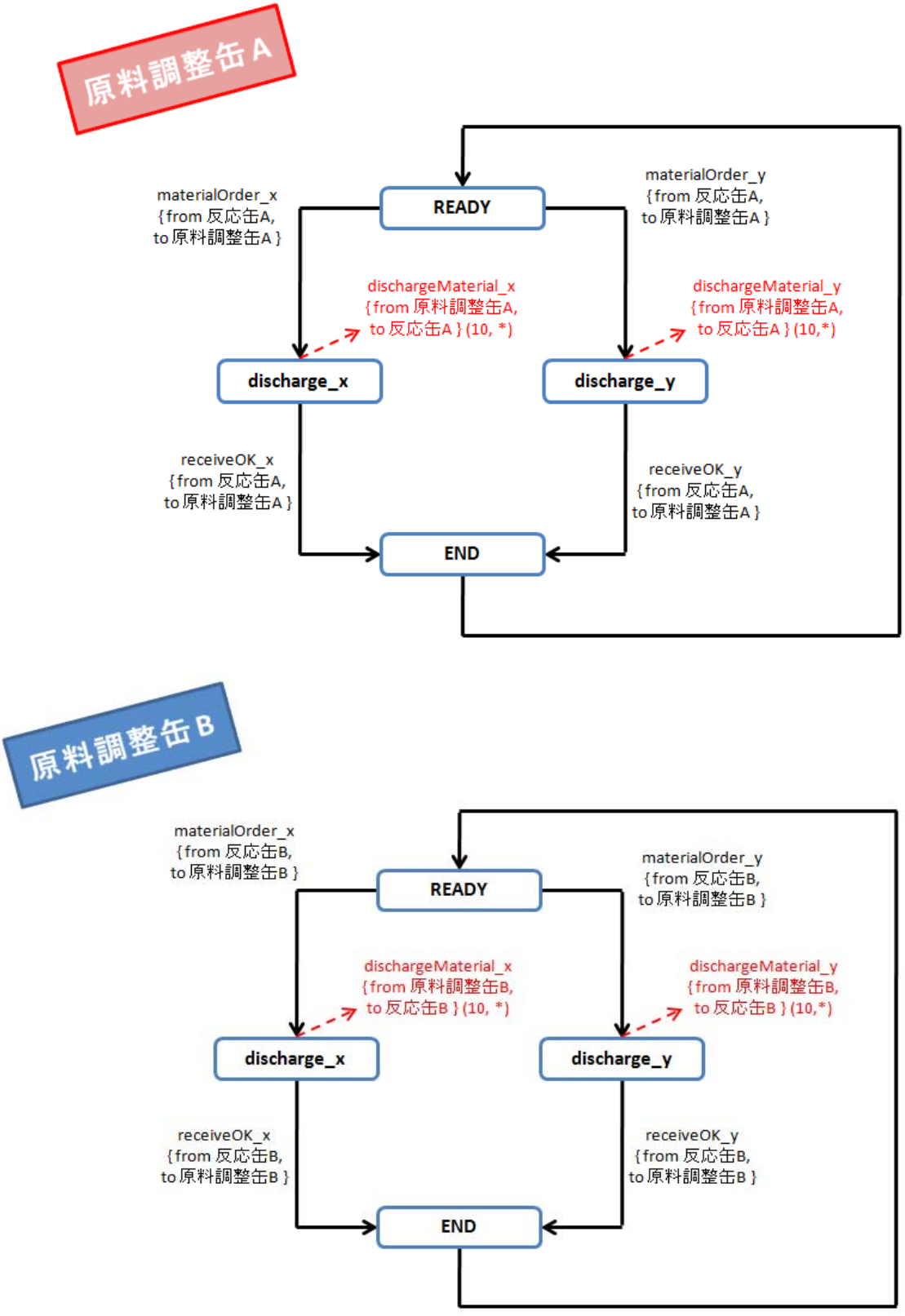
<ControlList>
<ControlListName> List1 </ControlListName>
<ControlListElement> ICB5-e-C1 C1-ARB6-IC 0 </ControlListElement>
<ControlListElement> ICB5-e-C1 C1-ARB6-IC 0 </ControlListElement>
<ControlListElement> ICA5-e-C1 C1-ARA6-IC 0 </ControlListElement>
<ControlListElement> ICA5-e-C1 C1-ARA6-IC 0 </ControlListElement>
<ControlListElement> T4B11-e-C1 C1-ARB12-T4 1 </ControlListElement>
<ControlListElement> T3B11-e-C1 C1-ARB12-T3 1 </ControlListElement>
<ControlListElement> ICA5-e-C1 C1-ARA6-IC 1 </ControlListElement>
<ControlListElement> T2A11-e-C1 C1-ARA12-T2 1 </ControlListElement>
<ControlListElement> T1A11-e-C1 C1-ARA12-T1 1 </ControlListElement>
<ControlListElement> ICA5-e-C1 C1-ARA6-IC 1 </ControlListElement>
<ControlListElement> T4A11-e-C1 C1-ARA12-T4 1 </ControlListElement>
<ControlListElement> T2A11-e-C1 C1-ARA12-T2 1 </ControlListElement>
</ControlList>

// ○ControlList2 (ターンテーブル位置設定順序)
<ControlList>
<ControlListName> List2 </ControlListName>
<ControlListElement> ARB6-e-C2 C2-TTB8-AR 0 </ControlListElement>
<ControlListElement> ARB6-e-C2 C2-TTB8-AR 0 </ControlListElement>
<ControlListElement> ARA6-e-C2 C2-TTA8-AR 0 </ControlListElement>
<ControlListElement> ARA6-e-C2 C2-TTA8-AR 0 </ControlListElement>
<ControlListElement> T4B11-e-C2 C2-TTB13-T4 1 </ControlListElement>
<ControlListElement> T3B11-e-C2 C2-TTB13-T3 1 </ControlListElement>
<ControlListElement> ARA6-e-C2 C2-TTA8-AR 1 </ControlListElement>
<ControlListElement> T2A11-e-C2 C2-TTA13-T2 1 </ControlListElement>
<ControlListElement> T1A11-e-C2 C2-TTA13-T1 1 </ControlListElement>
<ControlListElement> ARA6-e-C2 C2-TTA8-AR 1 </ControlListElement>
<ControlListElement> T4A11-e-C2 C2-TTA13-T4 1 </ControlListElement>
<ControlListElement> T2A11-e-C2 C2-TTA13-T2 1 </ControlListElement>
</ControlList>

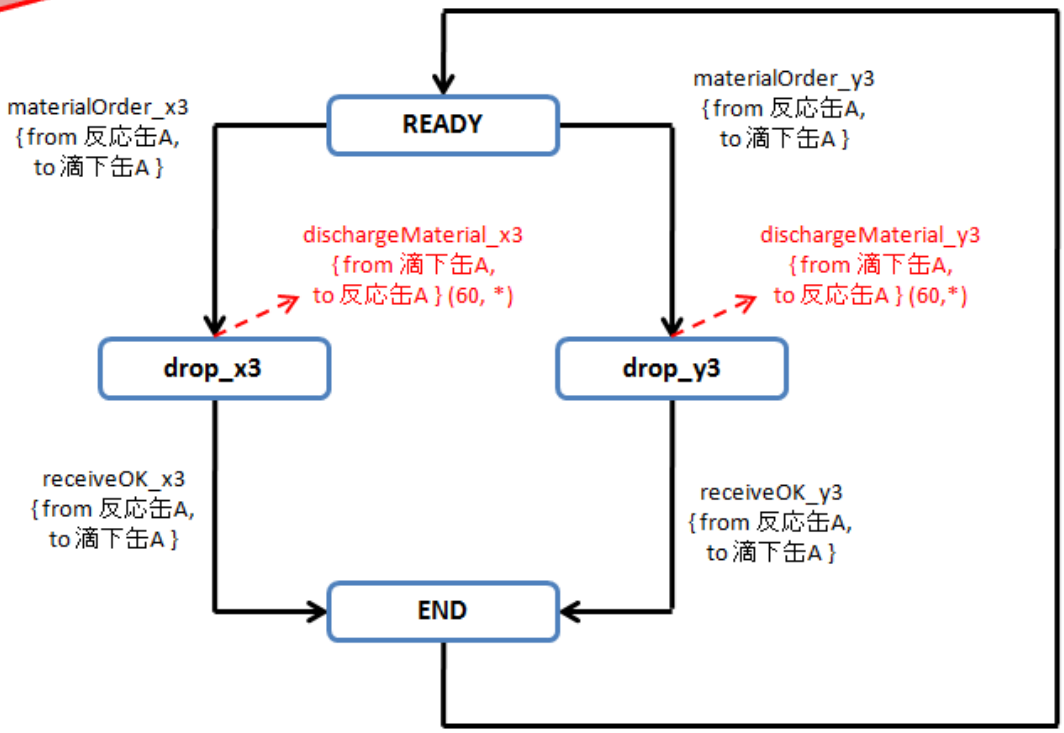
// ○ControlList (処理装置割当て)
<ControlList>
<ControlListName> List3 </ControlListName>
<ControlListElement> TTB8-s-C3 C3-T4B 0 </ControlListElement>
<ControlListElement> TTB8-s-C3 C3-T3B 0 </ControlListElement>
<ControlListElement> TTA8-s-C3 C3-T2A 0 </ControlListElement>
<ControlListElement> TTA8-s-C3 C3-T1A 0 </ControlListElement>
<ControlListElement> TTA8-s-C3 C3-T4A 1 </ControlListElement>
<ControlListElement> TTA8-s-C3 C3-T2A 1 </ControlListElement>
</ControlList>

// ○ControlList (ストッカー割り当て)
<ControlList>
<ControlListName> List4 </ControlListName>
<ControlListElement> OCB17-e-C4 C4-S3B 1 </ControlListElement>
<ControlListElement> OCB17-e-C4 C4-S4B 1 </ControlListElement>
<ControlListElement> OCA17-e-C4 C4-S2A 1 </ControlListElement>
<ControlListElement> OCA17-e-C4 C4-S1A 1 </ControlListElement>
<ControlListElement> OCA17-e-C4 C4-S4A 1 </ControlListElement>
<ControlListElement> OCA17-e-C4 C4-S2A 1 </ControlListElement>
</ControlList>

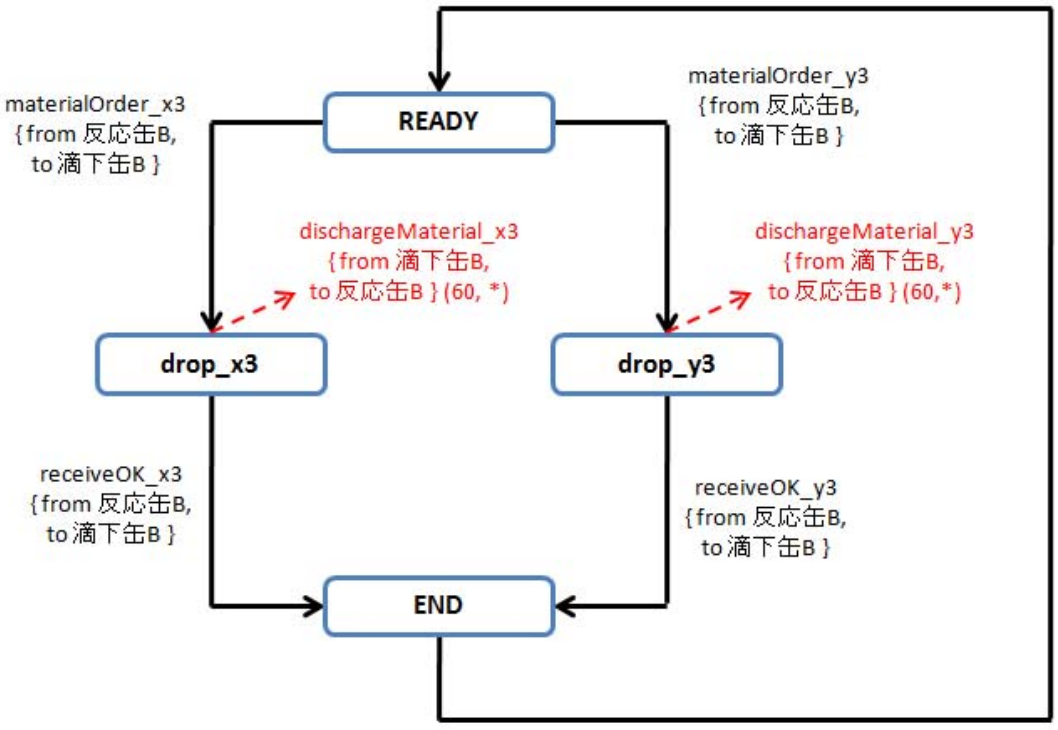
A2.1 バッチ式化学プラントの ETSC モデル



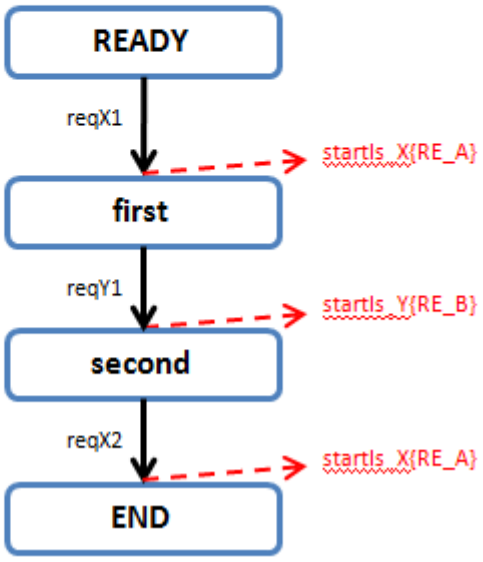
滴下缶A


READY
materialOrder_x3
{from 反応缶A,
to 滴下缶A }
materialOrder_y3
{from 反応缶A,
to 滴下缶A }
dischargeMaterial_x3
{from 滴下缶A,
to 反応缶A } (60, *)
dischargeMaterial_y3
{from 滴下缶A,
to 反応缶A } (60,*)
drop_x3
drop_y3
receiveOK_x3
{from 反応缶A,
to 滴下缶A }
receiveOK_y3
{from 反応缶A,
to 滴下缶A }
END

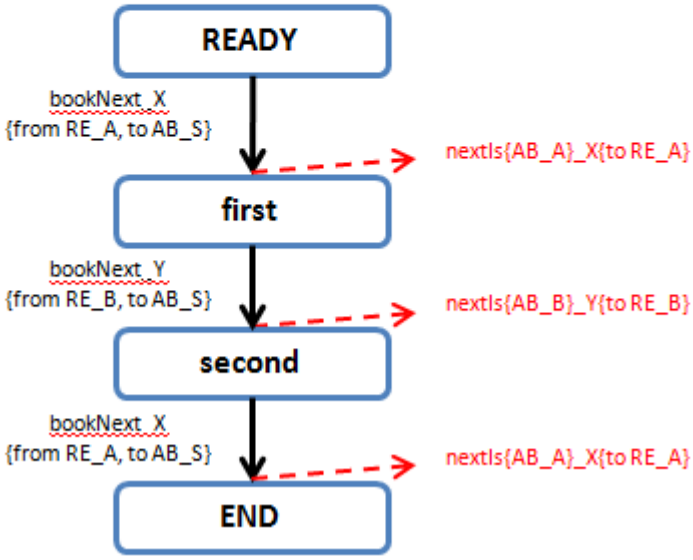
滴下缶B


READY
materialOrder_x3
{from 反応缶B,
to 滴下缶B }
materialOrder_y3
{from 反応缶B,
to 滴下缶B }
dischargeMaterial_x3
{from 滴下缶B,
to 反応缶B } (60, *)
dischargeMaterial_y3
{from 滴下缶B,
to 反応缶B } (60,*)
drop_x3
drop_y3
receiveOK_x3
{from 反応缶B,
to 滴下缶B }
receiveOK_y3
{from 反応缶B,
to 滴下缶B }
END

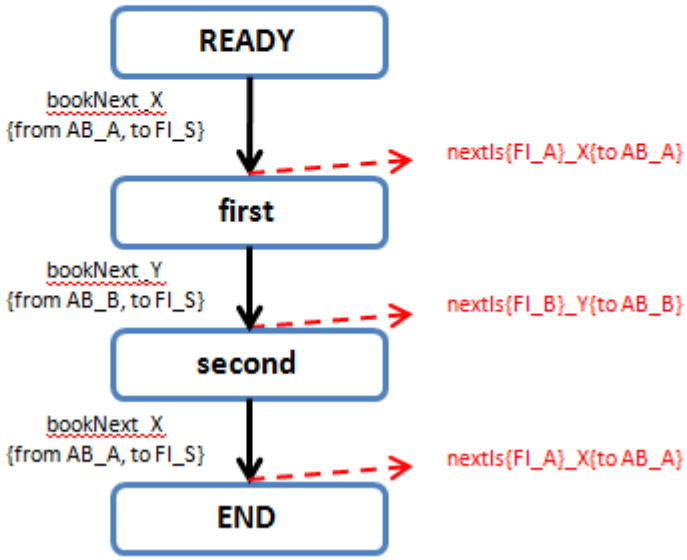
反応缶 Schedule



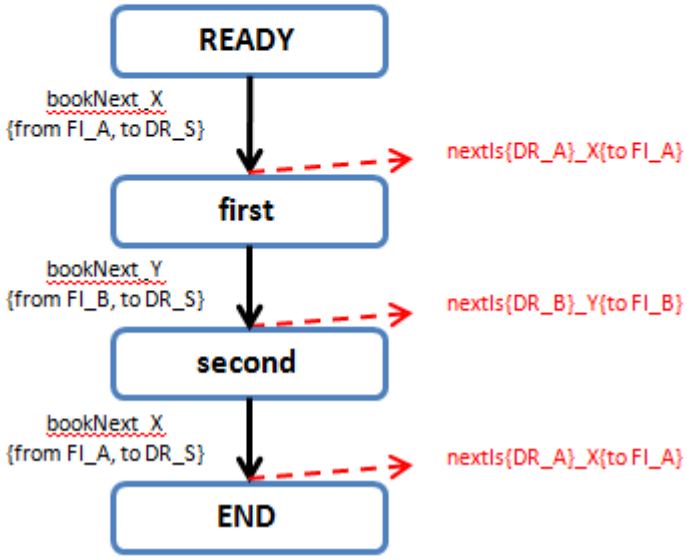
晶析缶 Schedule

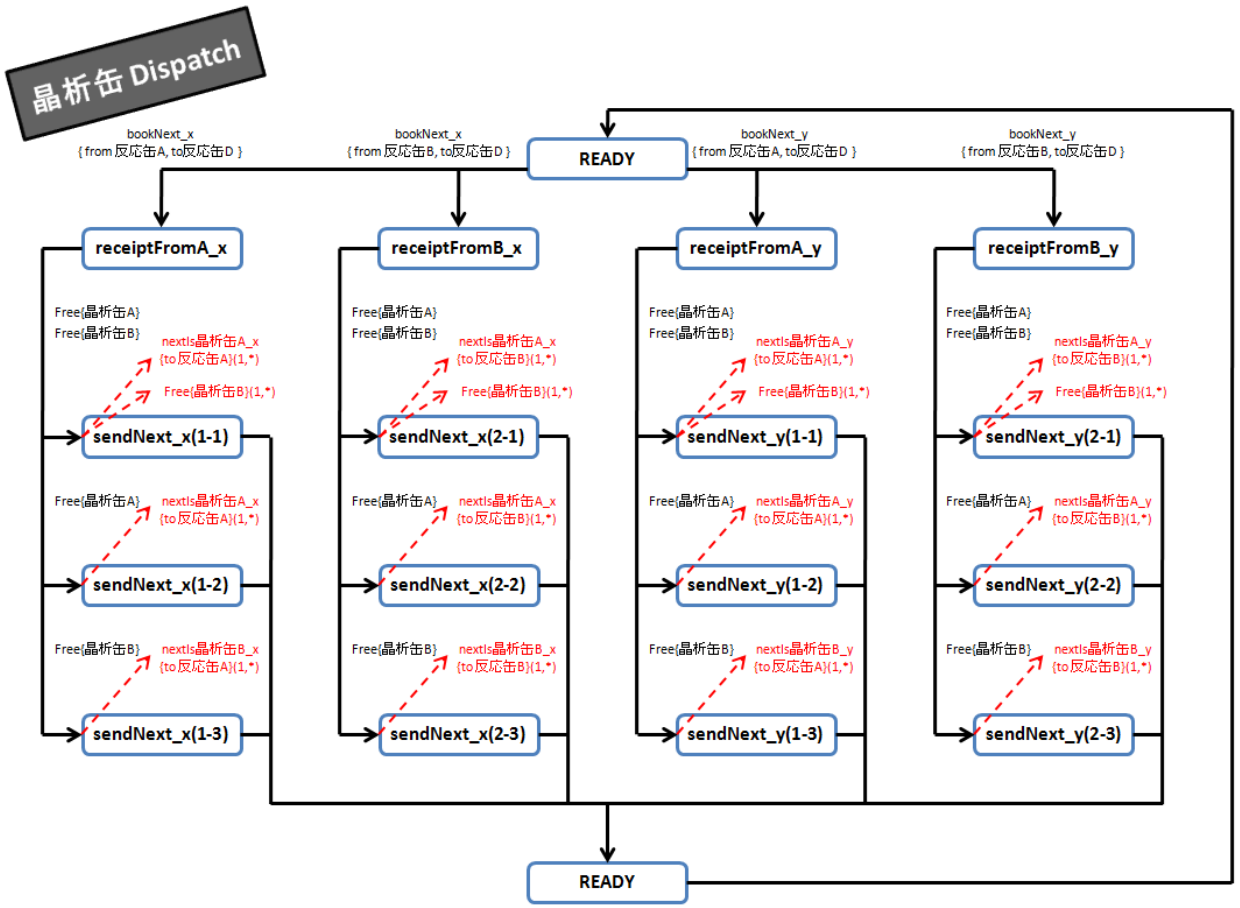


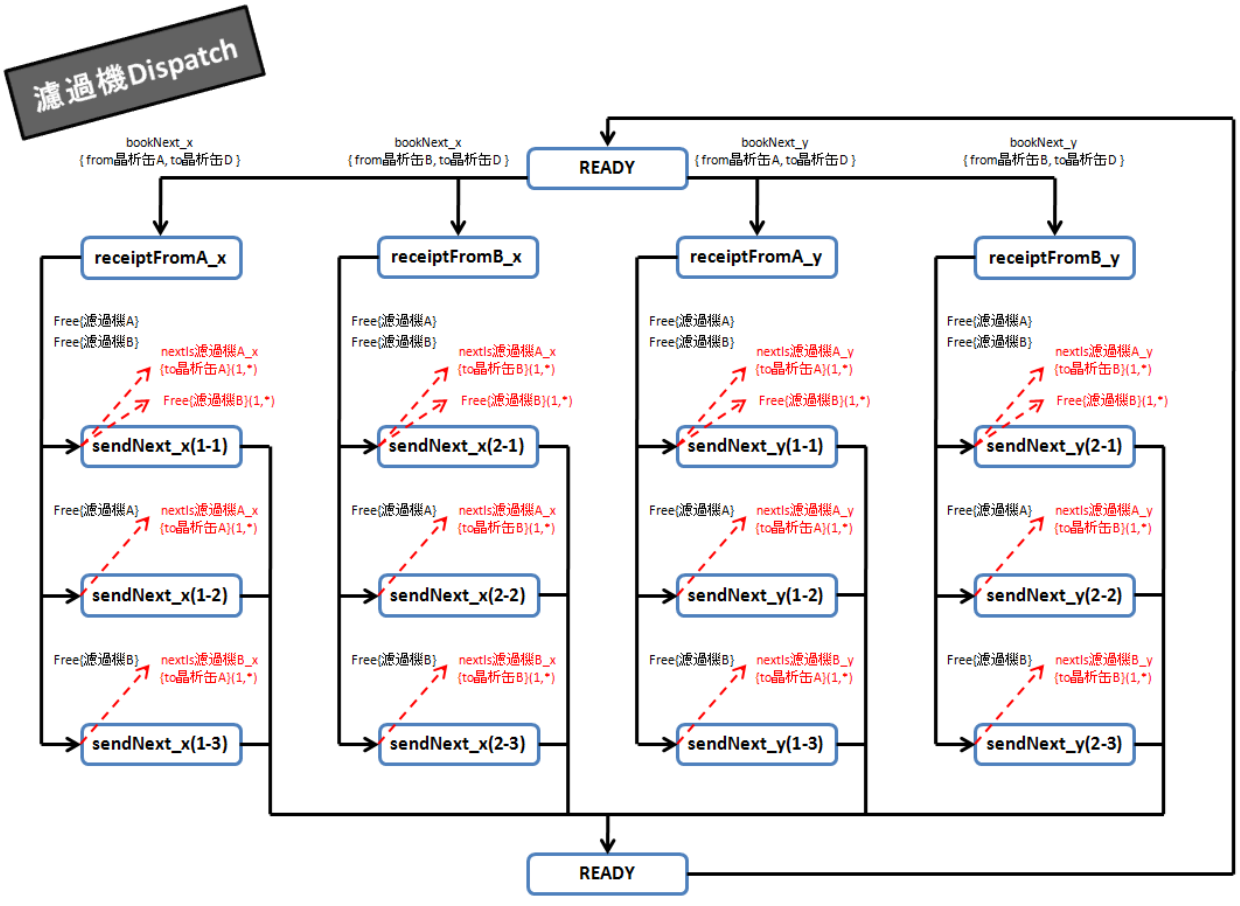
濾過機 Schedule



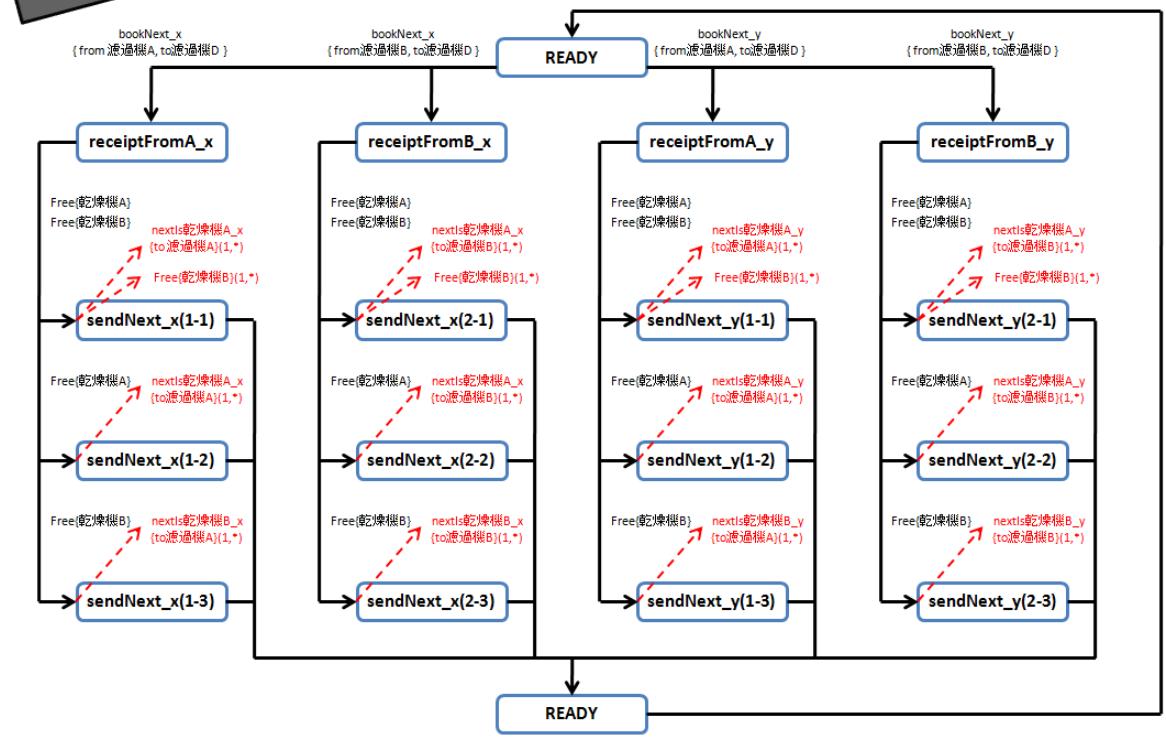
乾燥機 Schedule



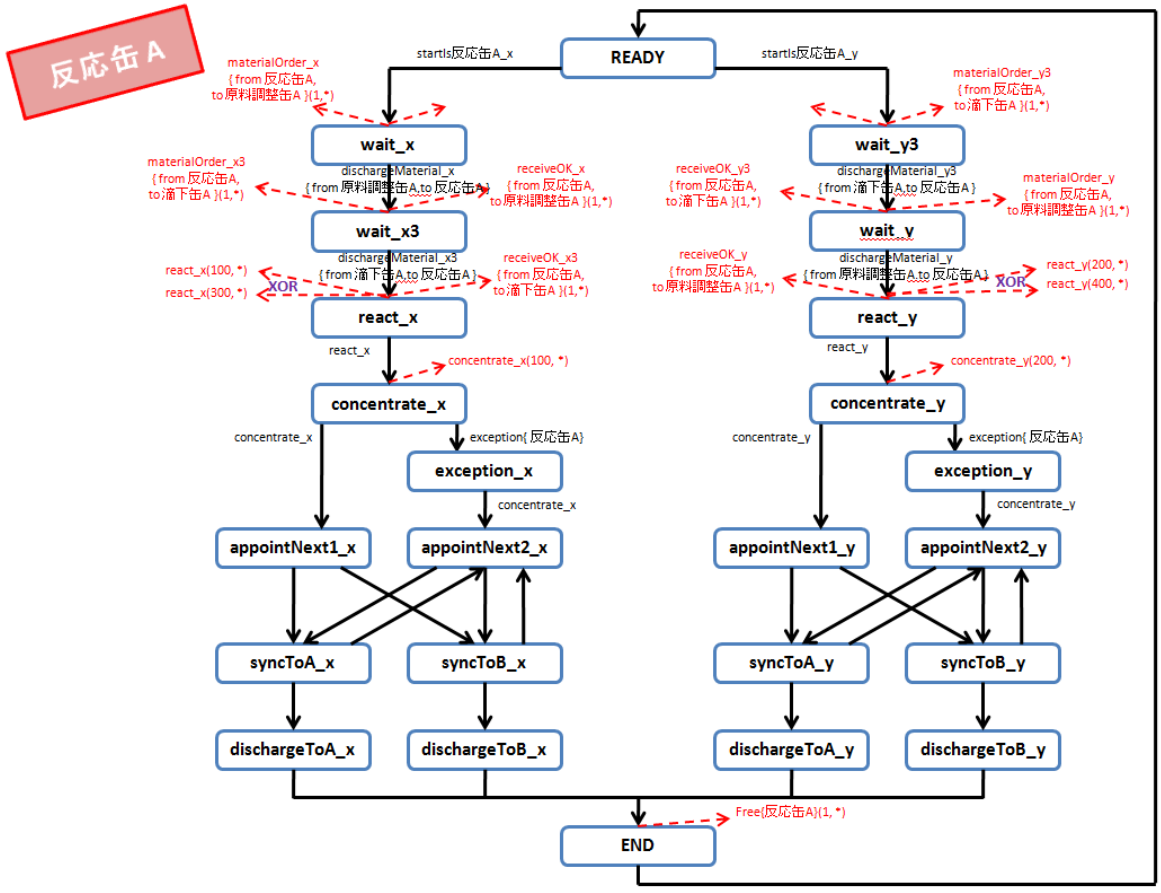

晶析缶 Dispatch
bookNext_x
{ from 反応缶A, to反応缶D }
bookNext_x
{ from 反応缶B, to反応缶D }
READY
bookNext_y
{ from 反応缶A, to反応缶D }
bookNext_y
{ from 反応缶B, to反応缶D }
receiptFromA_x
receiptFromB_x
receiptFromA_y
receiptFromB_y
Free{晶析缶A}
Free{晶析缶B}
nextIs晶析缶A_x
{to反応缶A}(1,*)
Free{晶析缶B}(1,*)
nextIs晶析缶A_x
{to反応缶B}(1,*)
nextIs晶析缶A_y
{to反応缶A}(1,*)
nextIs晶析缶A_y
{to反応缶B}(1,*)
sendNext_x(1-1)
sendNext_x(2-1)
sendNext_y(1-1)
sendNext_y(2-1)
Free{晶析缶A}
nextIs晶析缶A_x
{to反応缶A}(1,*)
nextIs晶析缶A_x
{to反応缶B}(1,*)
nextIs晶析缶A_y
{to反応缶A}(1,*)
nextIs晶析缶A_y
{to反応缶B}(1,*)
sendNext_x(1-2)
sendNext_x(2-2)
sendNext_y(1-2)
sendNext_y(2-2)
Free{晶析缶B}
nextIs晶析缶B_x
{to反応缶A}(1,*)
nextIs晶析缶B_x
{to反応缶B}(1,*)
nextIs晶析缶B_y
{to反応缶A}(1,*)
nextIs晶析缶B_y
{to反応缶B}(1,*)
sendNext_x(1-3)
sendNext_x(2-3)
sendNext_y(1-3)
sendNext_y(2-3)
READY

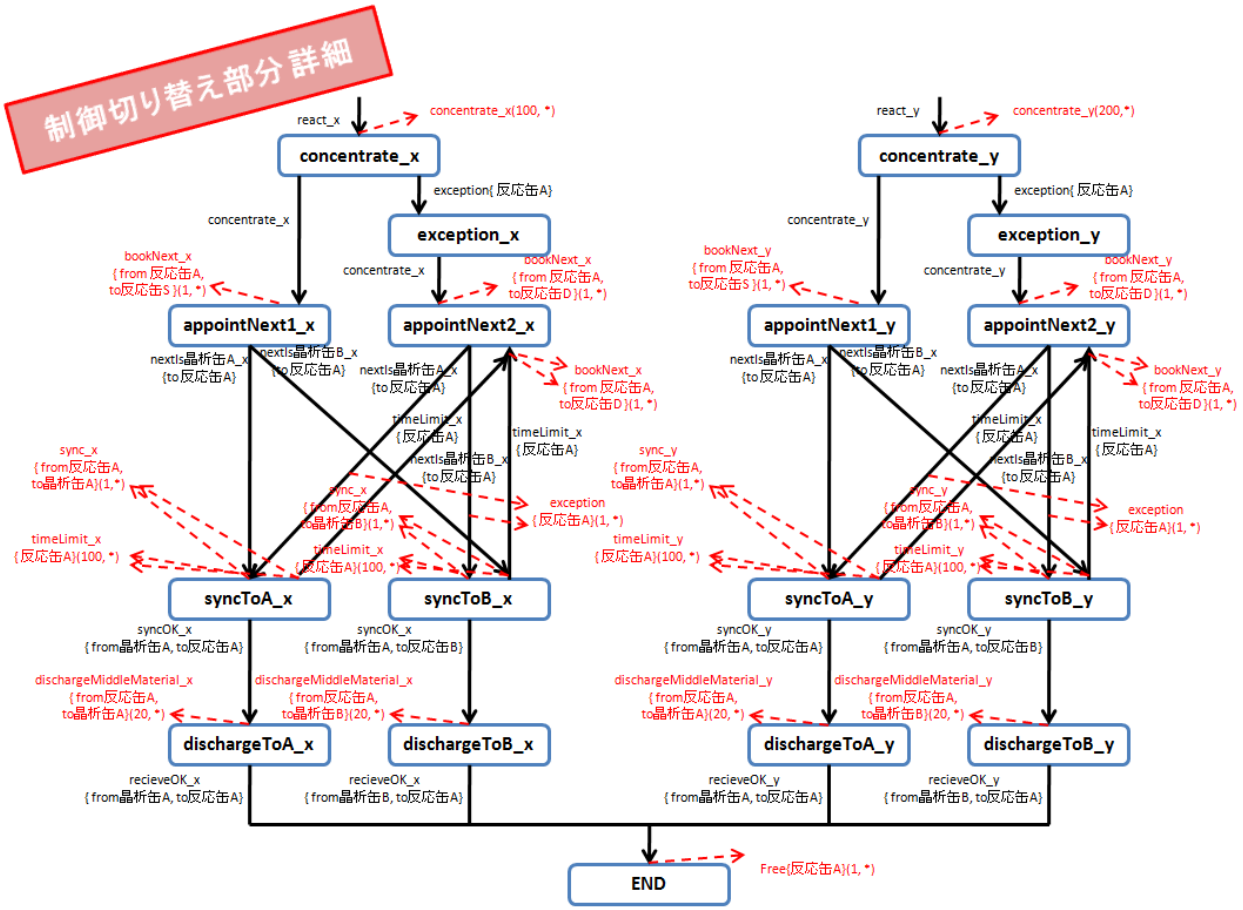

濾過機Dispatch
bookNext_x
{ from晶析缶A, to晶析缶D }
bookNext_x
{ from晶析缶B, to晶析缶D }
READY
bookNext_y
{ from晶析缶A, to晶析缶D }
bookNext_y
{ from晶析缶B, to晶析缶D }
receiptFromA_x
receiptFromB_x
receiptFromA_y
receiptFromB_y
Free{濾過機A}
Free{濾過機B}
nextIs濾過機A_x
{to晶析缶A}(1,*)
Free{濾過機B}(1,*)
nextIs濾過機A_x
{to晶析缶B}(1,*)
nextIs濾過機A_y
{to晶析缶A}(1,*)
nextIs濾過機A_y
{to晶析缶B}(1,*)
sendNext_x(1-1)
sendNext_x(2-1)
sendNext_y(1-1)
sendNext_y(2-1)
Free{濾過機A}
nextIs濾過機A_x
{to晶析缶A}(1,*)
nextIs濾過機A_x
{to晶析缶B}(1,*)
nextIs濾過機A_y
{to晶析缶A}(1,*)
nextIs濾過機A_y
{to晶析缶B}(1,*)
sendNext_x(1-2)
sendNext_x(2-2)
sendNext_y(1-2)
sendNext_y(2-2)
Free{濾過機B}
nextIs濾過機B_x
{to晶析缶A}(1,*)
nextIs濾過機B_x
{to晶析缶B}(1,*)
nextIs濾過機B_y
{to晶析缶A}(1,*)
nextIs濾過機B_y
{to晶析缶B}(1,*)
sendNext_x(1-3)
sendNext_x(2-3)
sendNext_y(1-3)
sendNext_y(2-3)
READY

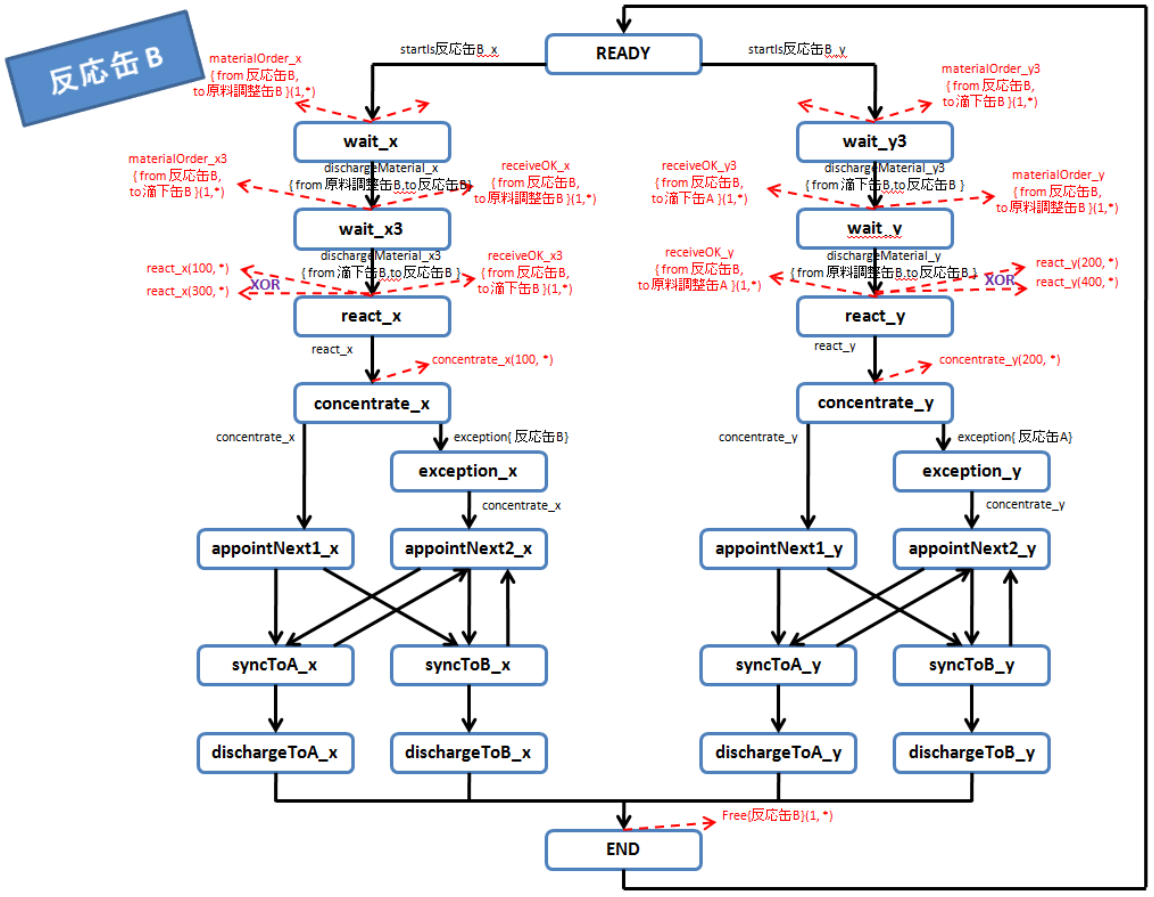
濾過機 Dispatch

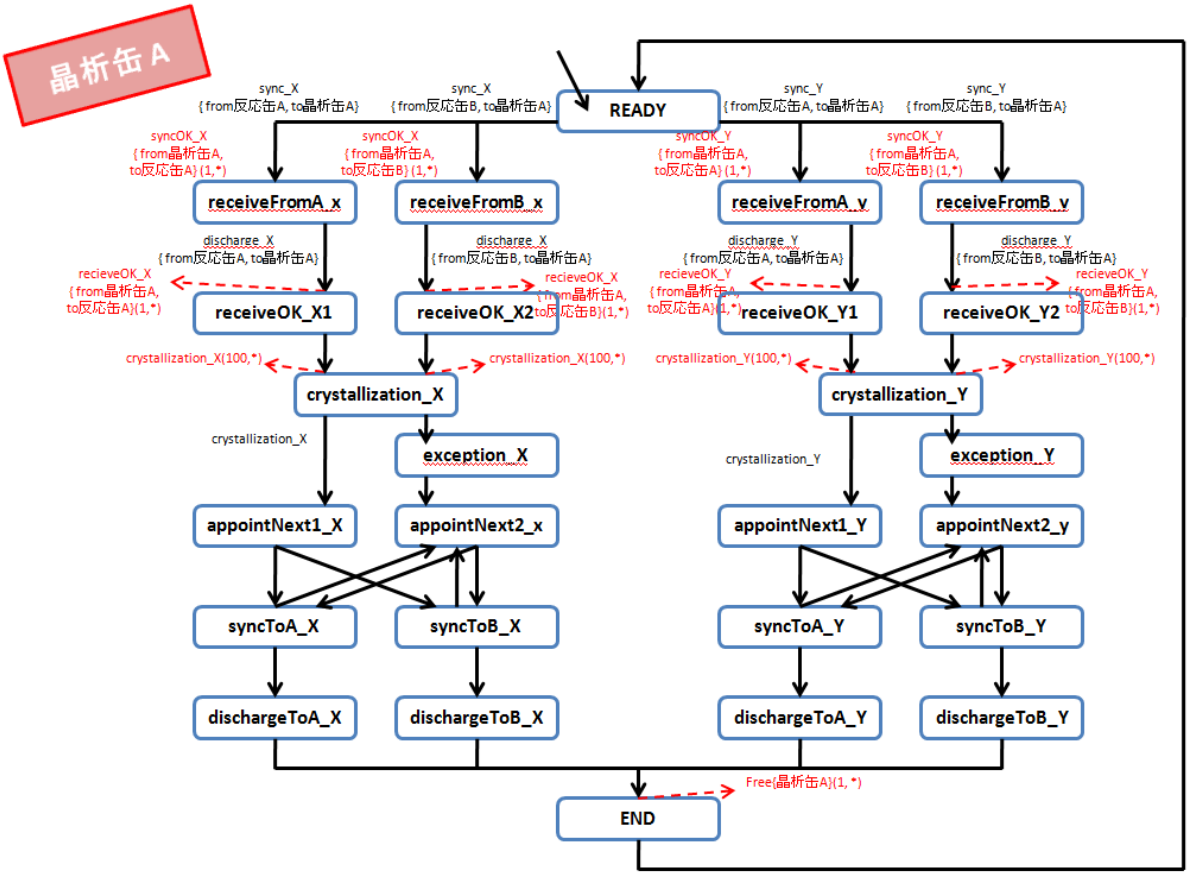


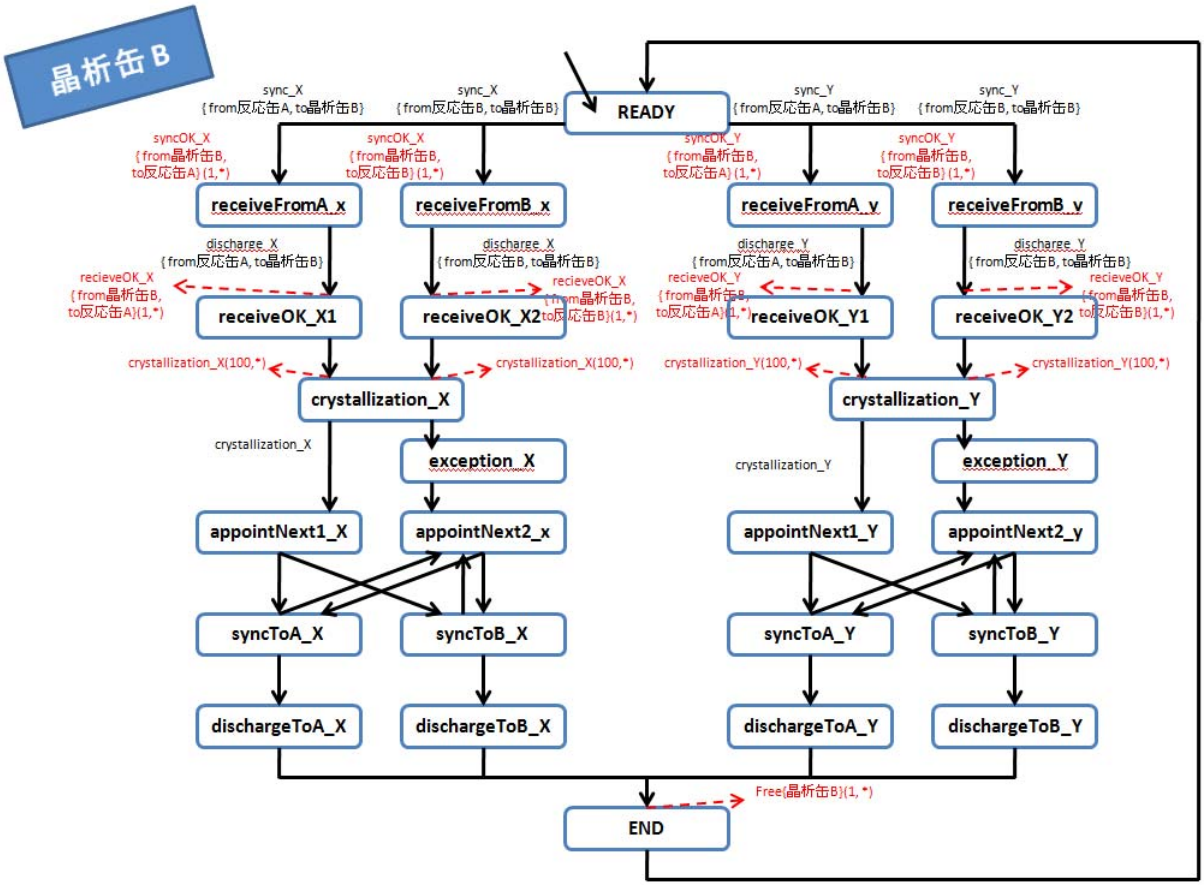
反応缶 A

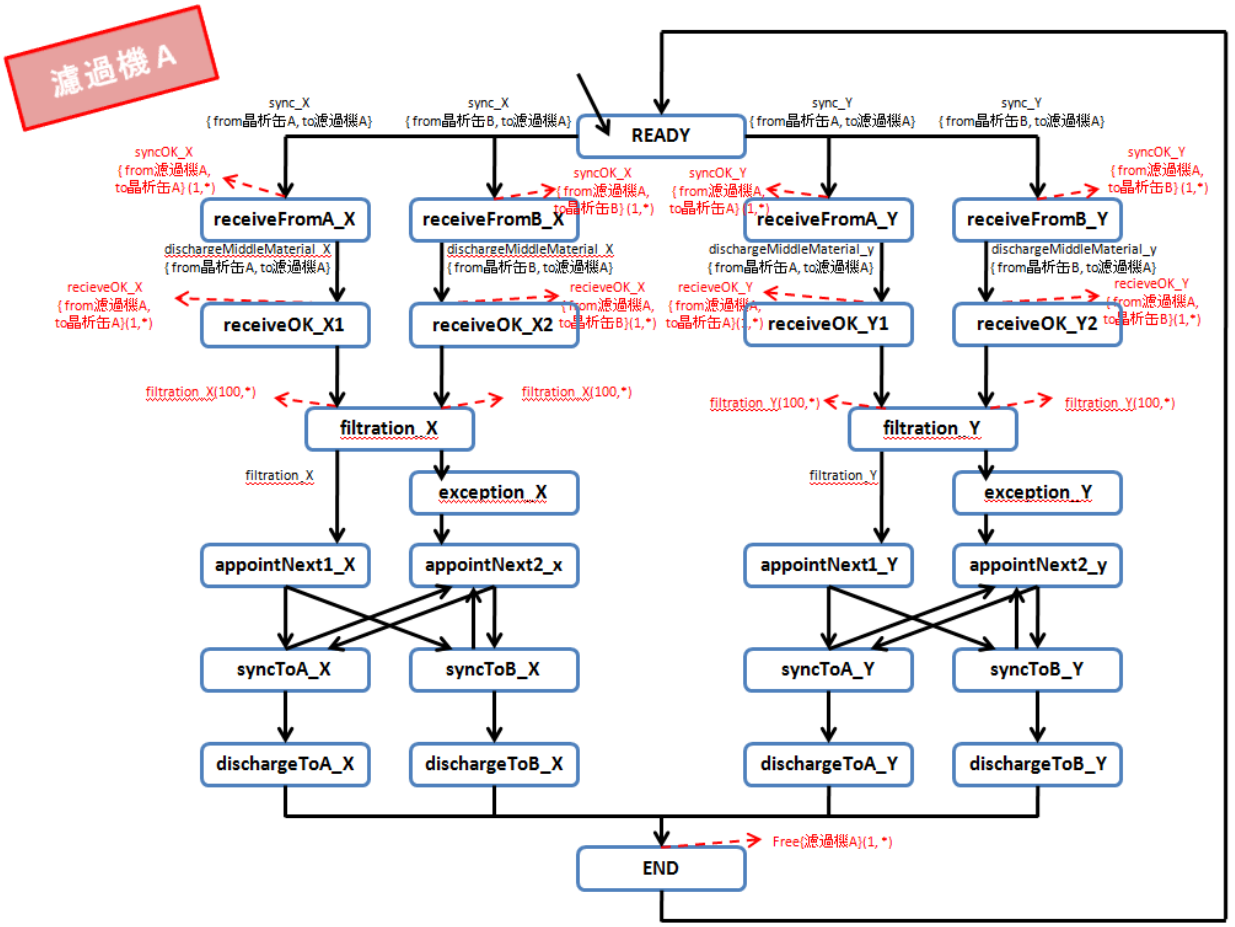


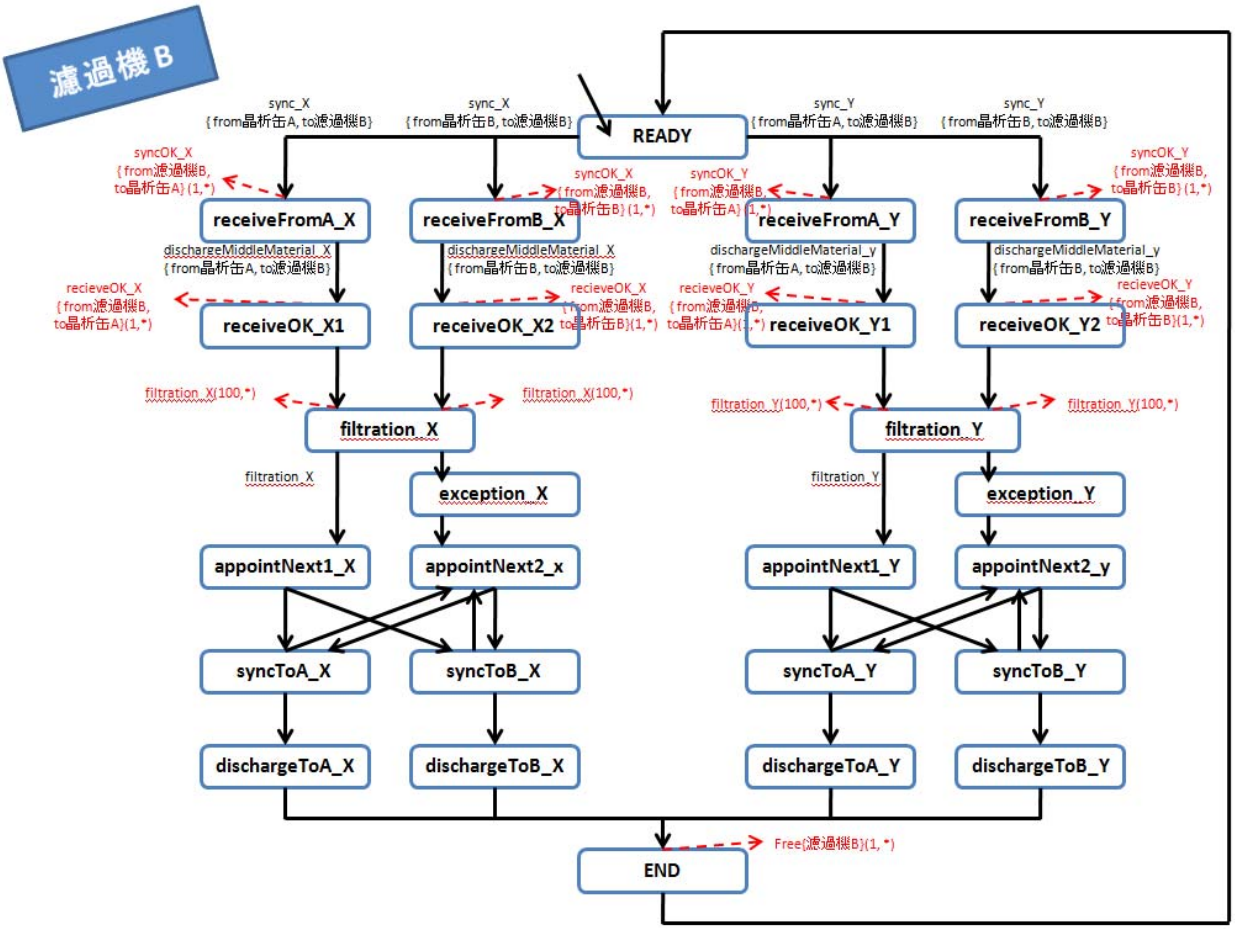

制御切り替え部分詳細
react_x
concentrate_x(100, *)
concentrate_x
exception{ 反応缶A}
exception_x
bookNext_x
{ from 反応缶A,
to反応缶S }(1, *)
bookNext_x
{ from 反応缶A,
to反応缶D }(1, *)
appointNext1_x
appointNext2_x
nextIs晶析缶A_x
{to反応缶A}
nextIs晶析缶B_x
{to反応缶A}
timeLimit_x
{反応缶A}
sync_x
{ from反応缶A,
to晶析缶A}(1,*)
sync_x
{ from反応缶A,
to晶析缶B}(1,*)
exception
{反応缶A}(1, *)
timeLimit_x
{反応缶A}(100, *)
syncToA_x
syncToB_x
syncOK_x
{ from晶析缶A, to反応缶A}
syncOK_x
{ from晶析缶A, to反応缶B}
dischargeMiddleMaterial_x
{ from反応缶A,
to晶析缶A}(20, *)
dischargeMiddleMaterial_x
{ from反応缶A,
to晶析缶B}(20, *)
dischargeToA_x
dischargeToB_x
recieveOK_x
{ from晶析缶A, to反応缶A}
recieveOK_x
{ from晶析缶B, to反応缶A}
END
Free{反応缶A}(1, *)
react_y
concentrate_y(200,*)
concentrate_y
exception{ 反応缶A}
exception_y
bookNext_y
{ from 反応缶A,
to反応缶S }(1, *)
bookNext_y
{ from 反応缶A,
to反応缶D }(1, *)
appointNext1_y
appointNext2_y
nextIs晶析缶A_x
{to反応缶A}
nextIs晶析缶B_x
{to反応缶A}
timeLimit_x
{反応缶A}
sync_y
{ from反応缶A,
to晶析缶A}(1,*)
sync_y
{ from反応缶A,
to晶析缶B}(1,*)
exception
{反応缶A}(1, *)
timeLimit_y
{反応缶A}(100, *)
syncToA_y
syncToB_y
syncOK_y
{ from晶析缶A, to反応缶A}
syncOK_y
{ from晶析缶A, to反応缶B}
dischargeMiddleMaterial_y
{ from反応缶A,
to晶析缶A}(20, *)
dischargeMiddleMaterial_y
{ from反応缶A,
to晶析缶B}(20, *)
dischargeToA_y
dischargeToB_y
recieveOK_y
{ from晶析缶A, to反応缶A}
recieveOK_y
{ from晶析缶B, to反応缶A}

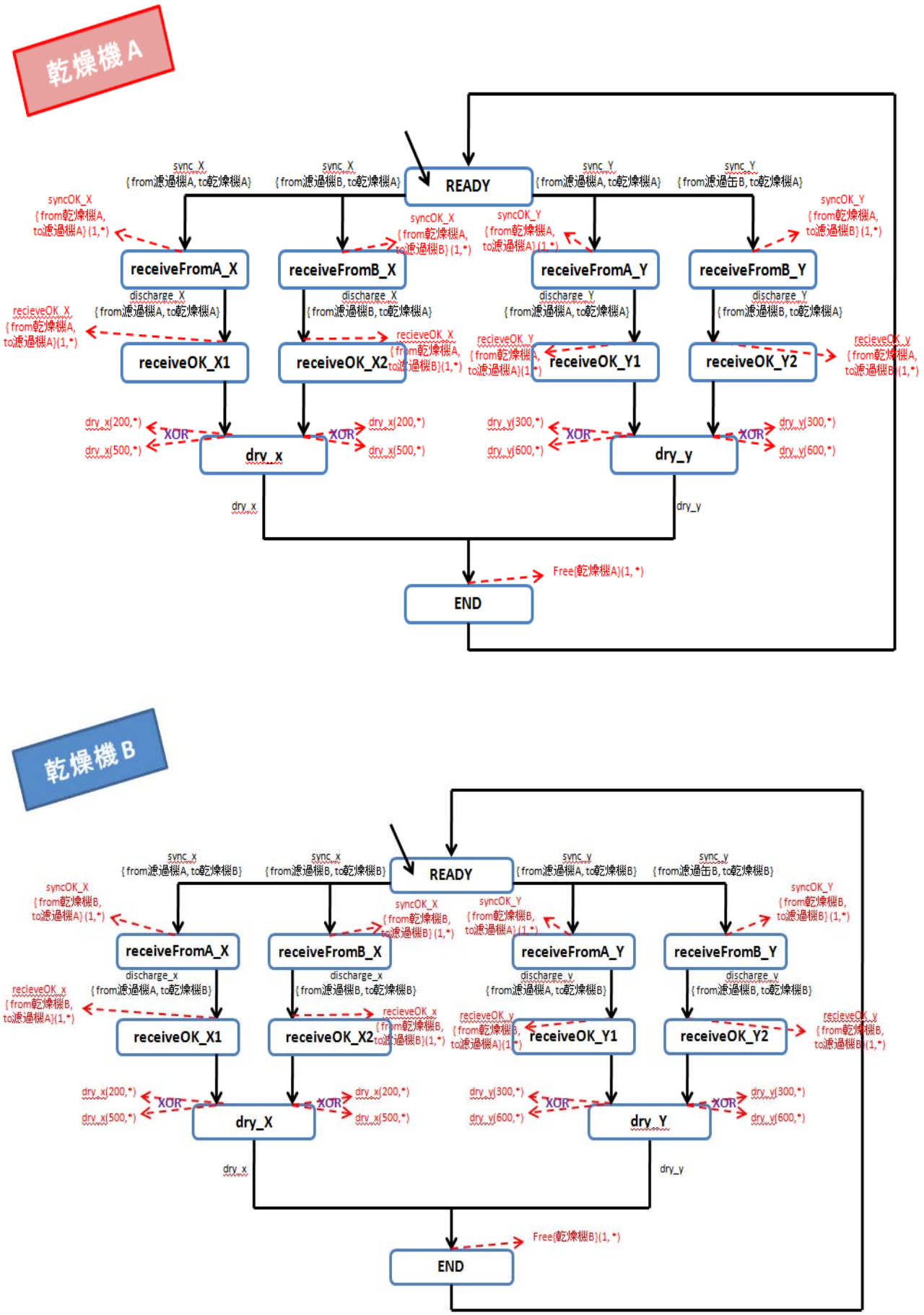

反応缶B
READY
startIs反応缶B_x
startIs反応缶B_y
materialOrder_x
{ from 反応缶B,
to原料調整缶B }(1,*)
wait_x
dischargeMaterial_x
{ from 原料調整缶B,to反応缶B}
materialOrder_x3
{ from 反応缶B,
to滴下缶B }(1,*)
receiveOK_x
{ from 反応缶B,
to原料調整缶B }(1,*)
wait_x3
dischargeMaterial_x3
{ from 滴下缶B,to反応缶B }
receiveOK_x3
{ from 反応缶B,
to滴下缶B }(1,*)
react_x(100, *)
react_x(300, *)
XOR
react_x
react_x
concentrate_x(100, *)
concentrate_x
exception{ 反応缶B}
exception_x
concentrate_x
appointNext1_x
appointNext2_x
syncToA_x
syncToB_x
dischargeToA_x
dischargeToB_x
Free{反応缶B}(1, *)
END
materialOrder_y3
{ from 反応缶B,
to滴下缶B }(1,*)
wait_y3
dischargeMaterial_y3
{ from 滴下缶B,to反応缶B }
receiveOK_y3
{ from 反応缶B,
to滴下缶A }(1,*)
materialOrder_y
{ from 反応缶B,
to原料調整缶B }(1,*)
wait_y
dischargeMaterial_y
{ from 原料調整缶B,to反応缶B }
receiveOK_y
{ from 反応缶B,
to原料調整缶A }(1,*)
react_y(200, *)
react_y(400, *)
XOR
react_y
react_y
concentrate_y(200, *)
concentrate_y
exception{ 反応缶A}
exception_y
concentrate_y
appointNext1_y
appointNext2_y
syncToA_y
syncToB_y
dischargeToA_y
dischargeToB_y


晶析缶A
sync_X
{from反応缶A, to晶析缶A}
sync_X
{from反応缶B, to晶析缶A}
READY
sync_Y
{from反応缶A, to晶析缶A}
sync_Y
{from反応缶B, to晶析缶A}
syncOK_X
{from晶析缶A, to反応缶A}(1,*)
syncOK_X
{from晶析缶A, to反応缶B}(1,*)
syncOK_Y
{from晶析缶A, to反応缶A}(1,*)
syncOK_Y
{from晶析缶A, to反応缶B}(1,*)
receiveFromA_x
receiveFromB_x
receiveFromA_y
receiveFromB_y
discharge_X
{from反応缶A, to晶析缶A}
discharge_X
{from反応缶B, to晶析缶A}
discharge_Y
{from反応缶A, to晶析缶A}
discharge_Y
{from反応缶B, to晶析缶A}
recieveOK_X
{from晶析缶A, to反応缶A}(1,*)
recieveOK_X
{from晶析缶A, to反応缶B}(1,*)
recieveOK_Y
{from晶析缶A, to反応缶A}(1,*)
recieveOK_Y
{from晶析缶A, to反応缶B}(1,*)
receiveOK_X1
receiveOK_X2
receiveOK_Y1
receiveOK_Y2
crystallization_X(100,*)
crystallization_X(100,*)
crystallization_Y(100,*)
crystallization_Y(100,*)
crystallization_X
crystallization_Y
crystallization_X
exception_X
crystallization_Y
exception_Y
appointNext1_X
appointNext2_x
appointNext1_Y
appointNext2_y
syncToA_X
syncToB_X
syncToA_Y
syncToB_Y
dischargeToA_X
dischargeToB_X
dischargeToA_Y
dischargeToB_Y
Free{晶析缶A}(1,*)
END


晶析缶B
sync_X
{from反応缶A, to晶析缶B}
sync_X
{from反応缶B, to晶析缶B}
READY
sync_Y
{from反応缶A, to晶析缶B}
sync_Y
{from反応缶B, to晶析缶B}
syncOK_X
{from晶析缶B, to反応缶A}(1,*)
syncOK_X
{from晶析缶B, to反応缶B}(1,*)
syncOK_Y
{from晶析缶B, to反応缶A}(1,*)
syncOK_Y
{from晶析缶B, to反応缶B}(1,*)
receiveFromA_x
receiveFromB_x
receiveFromA_y
receiveFromB_y
discharge_X
{from反応缶A, to晶析缶B}
discharge_X
{from反応缶B, to晶析缶B}
discharge_Y
{from反応缶A, to晶析缶B}
discharge_Y
{from反応缶B, to晶析缶B}
recieveOK_X
{from晶析缶B, to反応缶A}(1,*)
recieveOK_X
{from晶析缶B, to反応缶B}(1,*)
recieveOK_Y
{from晶析缶B, to反応缶A}(1,*)
recieveOK_Y
{from晶析缶B, to反応缶B}(1,*)
receiveOK_X1
receiveOK_X2
receiveOK_Y1
receiveOK_Y2
crystallization_X(100,*)
crystallization_X(100,*)
crystallization_Y(100,*)
crystallization_Y(100,*)
crystallization_X
crystallization_Y
crystallization_X
exception_X
crystallization_Y
exception_Y
appointNext1_X
appointNext2_x
appointNext1_Y
appointNext2_y
syncToA_X
syncToB_X
syncToA_Y
syncToB_Y
dischargeToA_X
dischargeToB_X
dischargeToA_Y
dischargeToB_Y
Free{晶析缶B}(1,*)
END


濾過機A
sync_X
{from晶析缶A, to濾過機A}
sync_X
{from晶析缶B, to濾過機A}
READY
sync_Y
{from晶析缶A, to濾過機A}
sync_Y
{from晶析缶B, to濾過機A}
syncOK_X
{from濾過機A, to晶析缶A}(1,*)
syncOK_X
{from濾過機A, to晶析缶B}(1,*)
syncOK_Y
{from濾過機A, to晶析缶A}(1,*)
syncOK_Y
{from濾過機A, to晶析缶B}(1,*)
receiveFromA_X
receiveFromB_X
receiveFromA_Y
receiveFromB_Y
dischargeMiddleMaterial_X
{from晶析缶A, to濾過機A}
dischargeMiddleMaterial_X
{from晶析缶B, to濾過機A}
dischargeMiddleMaterial_y
{from晶析缶A, to濾過機A}
dischargeMiddleMaterial_y
{from晶析缶B, to濾過機A}
recieveOK_X
{from濾過機A, to晶析缶A}(1,*)
recieveOK_X
{from濾過機A, to晶析缶B}(1,*)
recieveOK_Y
{from濾過機A, to晶析缶A}(1,*)
recieveOK_Y
{from濾過機A, to晶析缶B}(1,*)
receiveOK_X1
receiveOK_X2
receiveOK_Y1
receiveOK_Y2
filtration_X(100,*)
filtration_X(100,*)
filtration_Y(100,*)
filtration_Y(100,*)
filtration_X
filtration_Y
filtration_X
exception_X
filtration_Y
exception_Y
appointNext1_X
appointNext2_x
appointNext1_Y
appointNext2_y
syncToA_X
syncToB_X
syncToA_Y
syncToB_Y
dischargeToA_X
dischargeToB_X
dischargeToA_Y
dischargeToB_Y
Free(濾過機A)(1, *)
END


濾過機B
sync_X
{from晶析缶A, to濾過機B}
sync_X
{from晶析缶B, to濾過機B}
READY
sync_Y
{from晶析缶A, to濾過機B}
sync_Y
{from晶析缶B, to濾過機B}
syncOK_X
{from濾過機B, to晶析缶A}(1,*)
syncOK_X
{from濾過機B, to晶析缶B}(1,*)
syncOK_Y
{from濾過機B, to晶析缶A}(1,*)
syncOK_Y
{from濾過機B, to晶析缶B}(1,*)
receiveFromA_X
receiveFromB_X
receiveFromA_Y
receiveFromB_Y
dischargeMiddleMaterial_X
{from晶析缶A, to濾過機B}
dischargeMiddleMaterial_X
{from晶析缶B, to濾過機B}
dischargeMiddleMaterial_y
{from晶析缶A, to濾過機B}
dischargeMiddleMaterial_y
{from晶析缶B, to濾過機B}
recieveOK_X
{from濾過機B, to晶析缶A}(1,*)
recieveOK_X
{from濾過機B, to晶析缶B}(1,*)
recieveOK_Y
{from濾過機B, to晶析缶A}(1,*)
recieveOK_Y
{from濾過機B, to晶析缶B}(1,*)
receiveOK_X1
receiveOK_X2
receiveOK_Y1
receiveOK_Y2
filtration_X(100,*)
filtration_X(100,*)
filtration_Y(100,*)
filtration_Y(100,*)
filtration_X
filtration_Y
filtration_X
exception_X
filtration_Y
exception_Y
appointNext1_X
appointNext2_x
appointNext1_Y
appointNext2_y
syncToA_X
syncToB_X
syncToA_Y
syncToB_Y
dischargeToA_X
dischargeToB_X
dischargeToA_Y
dischargeToB_Y
Free(濾過機B)(1, *)
END


乾燥機A
READY
sync_X {from:濾過機A, to:乾燥機A}
sync_X {from:濾過機B, to:乾燥機A}
sync_Y {from:濾過機A, to:乾燥機A}
sync_Y {from:濾過缶B, to:乾燥機A}
syncOK_X {from:乾燥機A, to:濾過機A}(1,*)
syncOK_X {from:乾燥機A, to:濾過機B}(1,*)
syncOK_Y {from:乾燥機A, to:濾過機A}(1,*)
syncOK_Y {from:乾燥機A, to:濾過機B}(1,*)
receiveFromA_X
receiveFromB_X
receiveFromA_Y
receiveFromB_Y
discharge_X {from:濾過機A, to:乾燥機A}
discharge_X {from:濾過機B, to:乾燥機A}
discharge_Y {from:濾過機A, to:乾燥機A}
discharge_Y {from:濾過機B, to:乾燥機A}
recieveOK_X {from:乾燥機A, to:濾過機A}(1,*)
recieveOK_X {from:乾燥機A, to:濾過機B}(1,*)
recieveOK_Y {from:乾燥機A, to:濾過機A}(1,*)
recieveOK_Y {from:乾燥機A, to:濾過機B}(1,*)
receiveOK_X1
receiveOK_X2
receiveOK_Y1
receiveOK_Y2
dry_x{200,*)
dry_x{500,*)
XOR
dry_x
dry_y{300,*)
dry_y{600,*)
dry_y
Free{乾燥機A}(1,*)
END
乾燥機B
READY
sync_x {from:濾過機A, to:乾燥機B}
sync_x {from:濾過機B, to:乾燥機B}
sync_y {from:濾過機A, to:乾燥機B}
sync_y {from:濾過缶B, to:乾燥機B}
syncOK_X {from:乾燥機B, to:濾過機A}(1,*)
syncOK_X {from:乾燥機B, to:濾過機B}(1,*)
syncOK_Y {from:乾燥機B, to:濾過機A}(1,*)
syncOK_Y {from:乾燥機B, to:濾過機B}(1,*)
receiveFromA_X
receiveFromB_X
receiveFromA_Y
receiveFromB_Y
discharge_x {from:濾過機A, to:乾燥機B}
discharge_x {from:濾過機B, to:乾燥機B}
discharge_y {from:濾過機A, to:乾燥機B}
discharge_y {from:濾過機B, to:乾燥機B}
recieveOK_x {from:乾燥機B, to:濾過機A}(1,*)
recieveOK_x {from:乾燥機B, to:濾過機B}(1,*)
recieveOK_Y {from:乾燥機B, to:濾過機A}(1,*)
recieveOK_y {from:乾燥機B, to:濾過機B}(1,*)
receiveOK_X1
receiveOK_X2
receiveOK_Y1
receiveOK_Y2
dry_x{200,*)
dry_x{500,*)
XOR
dry_X
dry_y{300,*)
dry_y{600,*)
dry_Y
dry_x
dry_y
Free{乾燥機B}(1,*)
END

参考文献

1) 山根 智: 実時間並行ソフトウェアの仕様記述と検証, 情報処理学会誌 Vol.37 No.2,pp.188-203 (1996)

2) 山根 智: 時間ステートチャートに基づくリアルタイムシステム検証方式，情報処理学会誌 Vol.35 No.12,pp.2640-2650 (1994)

3) Clark E.M., Emerson E.A., Sistla, A.P.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Trans. On Programming Languages and Systems, Vol.8, No.2, pp.244-263 (1986)

4) 寺田 博文, 土屋 達弘,菊野 享,”記号モデル検査を用いたステートチャートの検証”,電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス pp.17-24 (1999)

5) Francis Schneider, Steve M. Easterbrook, John R. Callahan and Gerard J.: Validating Requirements for Fault Tolerant Systems using Model Checking, ICRE '98 Proceedings of the 3rd International Conference on Requirements Engineering, pp.4-13

6) Fred Kroger: Temporal Logic of Programs, Springer-Verlag New York (1987) ISBN: 3-540-17030-8

7) 松本，内平，本位田：時相論理とその応用，Information Processing Society of Japan, Vol.30 No.6

8) S. D. BROOKES: A Theory of Communicating Sequential Processes, Journal of the Association for Computing Machinery, Vol 31, No 3, July 1984, pp 560-599

9) Jan Grbowski: On Partial Language, Fundam. Inform. Vol.4-1 427-498 (1981)

10) 鈴木 橋爪 他 ： 条件／事象ネットの最小実現を求める方法に関する考察, SICE SSI DES51, pp.7-12, (2013)

11) 平石邦彦: ペトリネット理論 http://www.jaist.ac.jp/is/labs/hira-lab/old_home_page/Petrinet.pdf

12) 中田明夫: 時間オートマトンのモデル検査，電子情報通信学会論文誌 2009/5 Vol. J92-D No.5

13) Harel D, Pnueli A, Schmidt J.P., Sherman R.: On Formal Semantics of Statecharts, IEEE Logic in Computer Science, pp.54-64 (1987)

14) Valeriy Vyatkin: IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design, ISA (2007)

15) James.H.Christensen: Design patterns for systems engineering with 61499 (http://www.holobloc.com/)

16) Hagge, N, Wagner, B.: Applying the handler-based execution model to IEC 61499 basic and composite function blocks, Industrial Informatics, 2006 IEEE International Conference on

17) 宮澤以鋼，長尾達明: 機能安全ファンクションブロックの検証に関する考察

18) Minoru Nakamura: ＭＥＳの役割と動向,『化学装置』誌 2001 年 1 月号

19) 大河内暁男: 経営構想力 東京大学出版会 (1979)

20) 佐藤知一: 生産マネジメント ワンポイント講義 (2)
http://www2.odn.ne.jp/scheduling/SCM/Onepoin2.html#anchor12970

21) 佐藤知一: MES（製造実行システム）とは何か（タイム・コンサルタントの日誌から）
http://brevis.exblog.jp/11092461

22) 独立行政法人 日本学術振興会プロセスシステム工学 第 143 委員会ジャパン バッチ フォーラム（JBF）編：常設分科会 S88 入門 -バッチシステムをよりよくデザインするために (2004)

23) 佐藤知一: 生産システムーその目的と機能は何か http://brevis.exblog.jp/8418442/

24) 独立行政法人工業所有権総合情報館: ネットワーク管理システム (2004)
www.inpit.go.jp/blob/katsuyo/pdf/chart/fkikai09.pdf

25) 高塚佳代子,冨田重幸: ネットワーク型生産システムの分権協調的運用モデルの提案とその検証，計測自動制御学会システム・情報部門学術講演会(SSI2011)論文集, pp.319-324 (2011)

26) 株式会社イニシア・コンサルティング「経営学講座」-理論から実践まで フォードシステム
http://www.initiaconsulting.co.jp/archives/management/3_03.html

27) 一般社団法人日本電気計測器工業会: プロセス計測制御機器の技術解説 (2010) tech.jemima.or.jp/

28) Takatsuka,Tomita: On a Formal Method for Modeling Discrete Production Systems based on Process-Network Architecture --With application to modeling of FA experiment device, Proc. of the International Symposium on Advanced Control of Industrial Processes 2002 (AdCONIP02), pp.281-286

29) Takatsuka,Tomita: On Model-based Approach for Developing Control Systems of Event-Driven Manufacturing Systems, Proc. of the 4th IEEE International Conference on Industrial Informatics, pp.699-706 (2006)

30) 高塚佳代子,冨田重幸: 離散型生産システムの挙動モデルにおける要制御部位の検出手法，計測自動制御学会システム・情報部門第 45 回離散事象システム研究会講演論文集, pp.1-6 (2009)

31) 高塚佳代子,冨田重幸: プロセスネットワークに基づく離散事象システムのモデル化手法，計測自動制御学会システム・情報部門第 48 回離散事象システム研究会講演論文集, pp.1-6 (2010)

32) 高塚佳代子、藤本政徳、冨田重幸: 離散事象システムの検証性質記述の操作性規範に基づく段階的作成法について，計測自動制御学会システム･情報部門第 49 回離散事象システム研究会講演論文集, pp.53-58 (2011)

33) 高塚佳代子、新名善久、冨田重幸: 離散型生産システムの検証における UPPAAL の適応範囲とモデル化の方法に関する検討（－拡張時間ステートチャートベースの検証手法との比較－）, 計測自動制御学会システム・情報部門第 50 回離散事象システム研究会講演論文集, pp.51-56 (2011)

34) 高塚佳代子、坂井雄志、山場久明、冨田重幸: "Matrix-Based Discrete-Event System Controller"の ETSC 挙動モデルを持つ対象への適用のための拡張, 計測自動制御学会システム・情報部門第 51 回離散事象システム研究会講演論文集, pp.77-84 (2012)

35) JBF/WG1 (http://jbf.pse143.org/wg.html) の研究会資料

36) 高塚, 舩場, 石橋, 平林, 冨田 バッチプラントの動作検証のためのモデルと検証アルゴリズムの提案 -制御ルールの公平性検証への適応例- ,化工 秋季大会, S306(2001)

37) Takatsuka,Tomita: On Modeling and Algorithm for verifying behavior of Discrete Parallel Production Systems, Proc. of the International Conference on Process Systems Engineering Asia (PSE ASIA), pp.277-282 (2005)

38) Takatsuka,Tomita: Modeling of Discrete Manufacturing Systems having multiple jobs for Verification by Model-Checking, Proc of the 8th IEEE International Conference on Industrial Informatics, pp.1136-1141 (2010)

39) 高塚佳代子,冨田重幸: 実行順序の追越を許した離散型生産システムの動作, 化学工学会第 75 年会発表予稿集, pp.183 (2010)

40) 高塚佳代子,冨田重幸: 複数の JOB の同時処理を行う離散型生産システムの動作検証手法の提案, 計測自動制御学会システム・情報部門第 47 回離散事象システム研究会講演論文集, pp.41-46 (2010)

41) 高塚佳代子、積俊行、山場久明、 岡崎直宣、冨田重幸: QE の活用による離散型生産システム制御系の適用限界とその評価方法, 計測自動制御学会システム・情報部門学術講演会(SSI2012)論文集, pp.531-536 (2012)

42) 高塚佳代子、山場久明、 岡崎直宣、冨田重幸: 離散型生産システムにおける処理時間の遅延リスクの評価方法について, 計測自動制御学会システム・情報部門第 53 回離散事象システム研究会講演論文集, pp.49-54 (2013)

43) 高塚佳代子, 冨田重幸: 複数ジョブを持つ離散型並列生産システムのための動作モデルの提案 (-バッチ式化学プラントの動作検証への適用), 計測自動制御学会論文集第 49 巻第 10 号, pp.901-910

44) 日本経営工学会マネージメント・エキスパート・システム研究部会編：戦略的生産管理システムとＡＩ的手法, 日刊工業新聞社, pp.129-146

45) 冨田, 山場, 大島: ツリー型工程を持つ多目的バッチプラントの知的運転支援システム -多品種同時生産での適応スケジューリングにおける経験則の活用-, 化学工学論文集, pp.740-749 (1991)

46) 日時伸哉: 遺伝的アルゴリズム GA 入門 ～遺伝的アルゴリズムを用いたテストケースの生成～ キャッツ組み込みソフトウェア研究所 CATS 2009, pp.1-12

47) 岡野: シミュレーテッド・アニーリング(SA 法)の概要と実験結果, (株)FFRI BLOG http://www.ffri.jp/blog/2013/07/2013-07-04.htm

48) 長尾, 中野, 日隈, 熊谷：FA システム用ペトリネットシミュレータの開発, 電子情報通信学会論文誌. A, 基礎・境界 J78-A(8), 901-909, 1995-08-25

49) Jose Mireles, Jr and Frank L. Lewis: Intelligent Material Handling: development and Implementation of a Matrix-Based Discrete-Event Controller, IEEE Transaction on Industrial electronics (2007)

50) Vincenzo Giordano, Jing Bing Zhang, David Naso, Frank Lewis, Alessandra Carbotti, Ng Tsong Jye: A matrix-based framework for combined supervisory and operational control of an industrial warehouse, INDIN2006 Proceeding pp.201-206

51) Basic Spin Manual (http://spinroot.com)

52) Gerard J.Holzman: THE SPIN MODEL CHECKER

53) Gerd Behrmann,Alexandre David,and Kim G. Larsen: A Tutorial on Uppaal", Department of Computer Science, Aalborg University,Denmark

54) Thomas Hune, Kim G. Larsen, Paul Pettersson: Guided Synthesis of Control Programs Using Uppaal, Nordic Journal of Computing (2001)

55) Torsten K. Iversen et al: Model-Checking Real-Time Control Programs -Verifying LEGO MINDSTORMS Systems Using UPPAAL, In Proc. of 12th Euromicro Conference on Real-Time Systems (2003)

56) Morten Laursen：Verifying Distributed LEGO RCX Programs Using UPPAAL, Department of Computer Science, Aalborg University, Denmark (1999)

57) K.L.McMillan "The SMV system for SMV version 2.5.4"

58) William Chan, Richard J. Anderson, Paul Beame, Steve Burns, Francesmary Modugno, David Notkin, Jon D. Reese, "Model Checking Large Software Specifications",IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 24, NO. 7, JULY 1998

59) V.G.KULKARNI： MARKOV AND MARKOV-REGENERATIVE PERT NETWORKS, Vol.34,No.5,September-October 1986

60) 増田士朗: 加工時間が確率変動する場合の遅延リスク評価に基づく最遅部品投入時刻の一計算法, SICE SSI DES49, pp.49-52, 2011

61) 橋爪進 他: 処理時間が確率変動する生産システムの処理完了時刻の推定, SICE SSI DES51, pp.11-14, 2012

62) Derek O'Connor: Exact and Approximate Distributions of Stochastic PERT Networks, Doctoral dissertation of University College, Dublin, 2007

63) Christopher W. Brown: An Overview of QEPCAD B: a Tool for Real Quantifier Elimination and Formula Simplification, Society for Symbolic and Algebraic Computation (2003)

64) 穴井他:計算機実代数幾何入門 数学セミナー2007～2009

65) PETER DORATO WEI YANG CHAOUKI ABDALLAH Robust: Multi-Objective Feedback Design by Quantifier Elimination, Symbolic Computation(1997)24,153-159

66) 諏訪晴彦・三道弘明,リアクティブスケジューリングにおけるスケジュール修正時期の判断基準に関する一考察（シミュレーションによる特性分析）,システム制御情報学会論文誌, Vol.15, No.7, pp.327-335 (2002)

67) 諏訪晴彦・三道弘明, 遅延タスク数を基準とした制御限界方策に基づくリアクティブスケジューリング, システム制御情報学会論文誌 Vol.16, No.11, pp.565-573 (2003)

68) Subanatarajan Subbiaha et al.: EngellaShort-Term Scheduling of Multi-Product Batch Plants with Sequence-Dependent Changeovers Using Timed Automata Models ESCAPE20 (2010)

69) Technical Report, PSE Research Group WS No.22, JSPS-RC143 (2002)

70) Heui-Seok Seo , Tadashi Araragi ,Yong Rae Kwon, Design and Analysis of Agent System by Extended Statecharts, IPSJ SIG Notes. ICS 2003(90) pp.145-152 27)

71) 山根智: 組込みシステムのフォーマルメソッドにおけるハイブリッドシステムの仕様記述と形式的検証, 電子通信情報学会,基礎・境界ソサイエティ, Fundamentals Review, Vol.2, No.1, pp.22-34 (2008)

72) Edward A. Lee, Thomas M. Parks: DATAFLOW PROCESS NETWORKS, Published in Proceedings of the IEEE, (May, 1995)

参考論文

25) 高塚佳代子,冨田重幸: ネットワーク型生産システムの分権協調的運用モデルの提案とその検証, 計測自動制御学会システム・情報部門学術講演会(SSI2011)論文集, pp.319-324 (2011)

28) Takatsuka,Tomita: On a Formal Method for Modeling Discrete Production Systems based on Process-Network Architecture --With application to modeling of FA experiment device, Proc. of the International Symposium on Advanced Control of Industrial Processes 2002 (AdCONIP02), pp.281-286

29) Takatsuka,Tomita: On Model-based Approach for Developing Control Systems of Event-Driven Manufacturing Systems, Proc. of the 4th IEEE International Conference on Industrial Informatics, pp.699-706 (2006)

30) 高塚佳代子,冨田重幸: 離散型生産システムの挙動モデルにおける要制御部位の検出手法, 計測自動制御学会システム・情報部門第 45 回離散事象システム研究会講演論文集, pp.1-6 (2009)

31) 高塚佳代子,冨田重幸: プロセスネットワークに基づく離散事象システムのモデル化手法, 計測自動制御学会システム・情報部門第 48 回離散事象システム研究会講演論文集, pp.1-6 (2010)

32) 高塚佳代子、藤本政徳、冨田重幸: 離散事象システムの検証性質記述の操作性規範に基づく段階的作成法について, 計測自動制御学会システム・情報部門第 49 回離散事象システム研究会講演論文集, pp.53-58 (2011)

33) 高塚佳代子、新名善久、冨田重幸: 離散型生産システムの検証における UPPAAL の適応範囲とモデル化の方法に関する検討（－拡張時間ステートチャートベースの検証手法との比較－）, 計測自動制御学会システム・情報部門第 50 回離散事象システム研究会講演論文集, pp.51-56 (2011)

34) 高塚佳代子、坂井雄志、山場久明、冨田重幸: "Matrix-Based Discrete-Event System Controller"の ETSC 挙動モデルを持つ対象への適用のための拡張, 計測自動制御学会システム・情報部門第 51 回離散事象システム研究会講演論文集, pp.77-84 (2012)

36) 高塚, 舩場, 石橋, 平林, 冨田 バッチプラントの動作検証のためのモデルと検証アルゴリズムの提案 -制御ルールの公平性検証への適応例- ,化工 秋季大会, S306(2001)

37) Takatsuka,Tomita: On Modeling and Algorithm for verifying behavior of Discrete Parallel Production Systems, Proc. of the International Conference on Process Systems Engineering Asia (PSE ASIA), pp.277-282 (2005)

38) Takatsuka,Tomita: Modeling of Discrete Manufacturing Systems having multiple jobs for Verification by Model-Checking, Proc of the 8th IEEE International Conference on Industrial Informatics, pp.1136-1141 (2010)

39) 高塚佳代子,冨田重幸: 実行順序の追越を許した離散型生産システムの動作, 化学工学会第 75 年会発表予稿集, pp.183 (2010)

40) 高塚佳代子,冨田重幸: 複数の JOB の同時処理を行う離散型生産システムの動作検証手法の提案, 計測自動制御学会システム・情報部門第 47 回離散事象システム研究会講演論文集, pp.41-46 (2010)

41) 高塚佳代子、積俊行、山場久明、 岡崎直宣、冨田重幸: QE の活用による離散型生産システム制御系の適用限界とその評価方法, 計測自動制御学会システム・情報部門学術講演会(SSI2012)論文集, pp.531-536 (2012)

42) 高塚佳代子、山場久明、 岡崎直宣、冨田重幸: 離散型生産システムにおける処理時間の遅延リスクの評価方法について, 計測自動制御学会システム・情報部門第 53 回離散事象システム研究会講演論文集, pp.49-54 (2013)

43) 高塚佳代子, 冨田重幸: 複数ジョブを持つ離散型並列生産システムのための動作モデルの提案 (-バッチ式化学プラントの動作検証への適用), 計測自動制御学会論文集第 49 巻第 10 号, pp.901-910

謝辞

本研究に際して、様々なご指導を頂きました主指導教員の冨田重幸教授に心より感謝申し上げます。また、本研究での実装面においてご指導ご協力を賜りました山場久昭助教に深く感謝申し上げます。また、本研究にご協力下さった冨田研究室の学生の皆様に心よりお礼申し上げます。更に、本論文執筆上のご指導を賜りました副指導教員の古谷博史教授、岡崎直宣教授に心より感謝申し上げます。最後に、家族に、なかでも、迷惑ばかりかけている母に、この場を借りて感謝したいと思います。