2007－SE－155 (3)
2007／3／22

# モデルレベルでの織り込みを実現するアスペクト指向設計環境

柳館 雄太　山形 晃人　横山 孝典
武蔵工業大学

本論文では，機能的側面と非機能的側面を分離してUML記述したモデルを織り込み，その結果をUML記述モデルとして確認できる，リアルタイムシステム向けのアスペクト指向設計環境を提案する．本手法では，既存のUMLエディタを用いてアスペクトの記述や織り込み結果の表示を行うことを可能とするため，UMLの構文に独自の拡張を加えないアスペクト記述法を提案するとともに，UMLの標準的な交換フォーマットであるXMIをベースにした織り込み機能を持つモデル・ウィーバを開発した．本モデルウィーバはリアルタイムシステムの振る舞いモデルを主な対象とし，UMLのシーケンス図とステートチャート図に対する織り込みが可能である．これにより，設計工程で非機能的側面の織り込み結果を確認しながら，インクリメンタルなモデル設計が可能になる．

# An Aspect-Oriented Modeling Environment for Model-Level Weaving

Yuta Yanagidate, Akito Yamagata and Takanori Yokoyama
Musashi Institute of Technology

The paper presents an aspect-oriented design environment for real-time systems. We can develop functional models and non-functional models independently, weave those models, and confirm the woven model with the design environment. We present an aspect modeling method without extending UML so that we represent aspect models using existing UML editors. We have developed a model weaver based on XMI format of UML. Sequence diagrams and state charts of UML are woven by the model weaver. The environment is useful for incremental real-time system modeling.

## 1　はじめに

一般に計算機システムに対する要求には，そのシステム本来の機能に関する機能的要求と，それ以外の非機能的要求とがある．特にリアルタイムシステムの場合には，リアルタイム性や信頼性等の非機能的要求が多く，ソフトウェアの中に機能的側面の処理と非機能的側面の処理が混在することが開発効率や再利用性の向上を難しくしている．したがって，機能的側面と非機能的側面を分離し，それぞれ独立に開発できる手法が求められている．

プログラムの複数の側面を分離して記述する方法として，アスペクト指向プログラミング[1]が注目されている．アスペクト指向は，複数のクラスに関わる処理（横断的関心事（Crosscutting Concern））を，クラスとは独立したアスペクトとして定義できる．アスペクト指向プログラミングをリアルタイムシステムに適用し，機能的側面から非機能的側面を分離して設計する方法も提案されている[2][3][4][5].

最近は，プログラミングのみでなく，アスペク

ト指向に基づく分析・設計の研究がなされている．これまでに提案されたアスペクト指向に基づくモデリング手法の多くは，UMLを拡張し，アスペクトを記述可能とするものである[6][7]．さらに，モデル記述のみでなく，モデルレベルでの織り込みを行うアスペクト指向開発環境も提案されている[8][9][10]．モデルレベルで織り込みを行うツールをモデルウィーバと呼ぶ．

リアルタイムシステムにおいても，モデルレベルで機能的側面と非機能的側面を分離して記述することで，設計の効率化や非機能的側面のモデルの再利用性向上が期待できる．しかし，リアルタイムシステムの設計では，単に複数の側面のモデルを独立に記述するのみでは十分ではない．タスクの設計や処理時間の予測などを行うには織り込み後のモデルを参照する必要がある．また，時間駆動やイベント駆動等の処理の駆動方法，タスク構成，同期や相互排除，ネットワーク通信等の非機能的側面の処理が多く存在するため，それらを一度に織り込むのではなく，ひとつずつインクリメンタルに織り込みながら設計を進める必要がある．

ところが，これまでに提案されているモデルウィーバの多くは，織り込み前のモデルを設計モデル，織り込み後のモデルを実装モデルとして，明確に分けて扱うのが普通である．コード自動生成を目的にしたものも多く，織り込み後の実装モデルを参照して追加設計を行うことは想定されていないため，織り込み結果を確認しながらのインクリメンタルな設計には適していない．また，多くは専用の環境であるため，既存のUMLエディタ環境への適用は容易ではない．

そこで我々の研究の目的は，リアルタイムシステム設計を主な対象に，インクリメンタルなアスペクト指向設計が可能な環境を実現することである．また導入を容易にするため，既存のUMLエディタをそのまま用いてアスペクトのモデルの記述や，織り込み後のモデルを確認できるようにしたい．

上記目的を達成するため本論文では，まず，UMLの構文に独自の拡張を加えないアスペクト記述を提案する．そして，UMLで記述した機能的側面のモデルと提案方法で非機能的側面を記述したアスペクトのモデルを入力し，UML記述された織り込み結果のモデルを出力するモデルウィーバを開発する．

リアルタイムシステム設計における非機能的側面の多くが振る舞いに関するものであるため，UMLのうちシーケンス図とステートチャート図を対象とする．また，モデルウィーバの入出力をUMLの標準的な交換フォーマットであるXMIをベースとすることで，既存の多くのUMLエディタ環境への適用を容易にする．これにより，設計工程で非機能的側面の織り込み結果を確認しながら，インクリメンタルなモデル設計が可能になる．

## 2　UMLによるアスペクト記述

### 2.1 アスペクトの記述要素

アスペクトにはジョインポイント，ポイントカット，アドバイスの3つの要素がある．ジョインポイントとはある処理を織り込むことのできるクラスの処理中の位置を意味し，例えば操作の呼び出し位置や属性の参照位置が挙げられる．ポイントカットはそのジョインポイントのうちどこを選択するかを指定する要素である．アドバイスとは，ポイントカットによって指定されたジョインポイントに織り込む処理である．このアドバイスには種類があり，ポイントカットで選択した位置の前に挿入するか，後に挿入するかなどを選択することができる．つまり，アスペクトの記述にはポイントカット，アドバイスの種類，アドバイス処理本体が必要となる．

アスペクト記述に含まれる上記要素を，UMLの文法をそのまま用いて記述できるようシーケンス図とステートチャート図を拡張する．以下2.2，2,3で，シーケンス図とステートチャート図について，記述法を具体的に述べる．

### 2.2 シーケンス図の記述法

本研究で提案するシーケンス図におけるアスペクト記法を図1に示す．まず，アスペクト名をオブジェクトアイコン内に記述する．その際，クラスとアスペクトを区別するため，ステレオタイプ"<<aspect>>"を記述する．

次にシーケンス図におけるジョインポイントについて説明する．シーケンス図においてはメッセージの根元と先端がジョインポイントである．メッセー

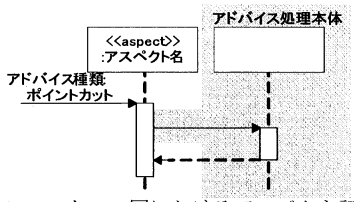

図1 シーケンス図におけるアスペクト記述

ジはあるオブジェクトの操作の起動やコンストラクタの起動を表現しているため，根元を呼び出し開始位置，先端を実行開始位置と考えることができる．また，リターンメッセージについては根元が実行終了位置，先端が呼び出し終了位置と考えられる．

任意のジョインポイントを指定するためにポイントカットとアドバイスの種類を記述する必要があるが，ここではAspectJ[11]の文法を適用する．記述する位置については，AspectJにおいてはポイントカットとアドバイスの種類の記述が操作名に相当することから，図1のようにメッセージ上に記述し，これによりアドバイス処理本体の呼び出し条件として扱うものとする．ここで，ジョインポイントを明示するためにリターンメッセージを記述することを条件とする．

具体的には呼び出し位置か実行位置かをcallポイントカットとexecutionポイントカットで選択し，開始位置か終了位置かをアドバイスの種類であるbeforeとafterによって選択する．また，指定した位置における処理の主体側と対象側のオブジェクトの種類を指定できるようthisポイントカットとtargetポイントカットを記述可能とする．また，その記述の際はワイルドカードやオペレータを使用できる．アドバイスの処理本体はクラスに織り込まれ統合される処理であり，アスペクト特有のものではない．よってアドバイス処理本体は従来のシーケンス図の記法をそのまま用いるものとする．

### 2.3 ステートチャート図の記述法

ステートチャート図におけるアスペクト記法を図2に示す．ステートチャート図では図2のようにコンポジット状態を用いてアスペクトを記述する．コンポジット状態の名前を記述する場所にステレオタイプ"<<aspect>>"を記述し，それがアスペクトであることを明示する．また，コンポジット状態の中に記述する状態遷移をアドバイス処理本体とする．

ステートチャート図においては，ジョインポイントをイベントと遷移先との接合点とし，ポイントカットによってジョインポイントを指定し織り込み箇所を特定する．ポイントカットとアドバイスの種類については<<aspect>>の後ろに書くものとする．ただし，この場合はAspectJの記法は用いず，遷移先とイベントのセットを指定することとする．具体的には，まず"<>"の中に遷移先の種類とイベントを記述する．例えば図2の例では状態とイベントのセットを指定している．その後で"()"中に実際の遷移先名とイベント名を指定する．複数指定する場合は"||"で区切ることとする．アドバイスの種類についてシーケンス図と同様にbeforeとafterを用いるものとし，それを"()"で囲まれた部分の直前に記述する．

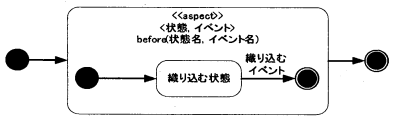

図2 ステートチャート図におけるアスペクト記述

## 3 モデルレベルの織り込み

### 3.1 モデルウィーバ

提案したアスペクト記法に従ってモデルの織り込みを行うモデルウィーバを開発した．本モデルウィーバは，多くのUMLエディタ環境に容易に適用できるよう，XMIフォーマットを入力して織り込みを行い，織り込み結果もXMIフォーマットとする．

ただし実際のUMLエディタでは，XMIのみでなく，画面表示のための描画情報が必要になるが，これについてはUMLエディタ依存にならざるを得ない．本研究では，扱えるUMLのバージョンは古いが，開発のための解析が容易なオープンソースのモデルエディタであるIIOSS[12]を対象とすることにした．IIOSSはUML1.1に準拠しているため，織り込み後のXMIファイルはUML1.1のメタデータに従った形で出力される．

開発したモデルウィーバにより織り込み処理の流れを図3に示す．ただし，具体的な処理について

は，シーケンス図とステートチャート図で異なるところがあるので，それぞれ3.2，3.3で説明する．なお，IIOSSではモデルの表示をPGML(Precision Graphics Markup Language)を用いて行っているため，図3の「描画情報の生成」ではPGMLファイルを生成している．この処理をモデルエディタにあわせて実装すれば，容易に他のモデルエディタ向けのモデルウィーバを実現できる．

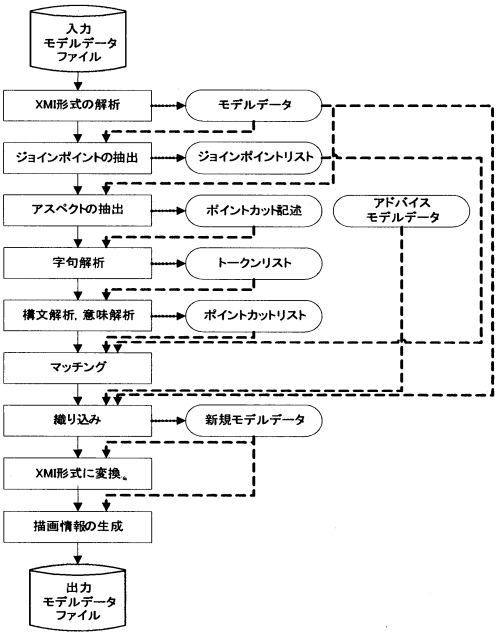


図3 モデルウィーバによる織り込み処理の流れ

### 3.2 シーケンス図の織り込み

シーケンス図の織り込み処理を，図3に沿って説明する．まずXMI形式のモデルデータファイルの解釈を行う．すなわち要素名や要素IDを抽出し，モデルデータを構築する．その際メッセージ要素についての情報を元に，メッセージの順序情報を生成する．同時にメッセージとリターンメッセージの対応付けも行う．この順序情報を元にジョインポイントリストを生成する．次にモデルの中からアスペクトが記述されているものを検索し，アドバイスの種類の記述を含むポイントカット記述と，アドバイス処理本体のモデルデータを抽出する．次にポイントカット記述の字句解析を行い，トークンリストにし，構文解析と意味解析を行いポイントカットリストを生成する．シーケンス図においてはAspectJの文法を採用しているため，修飾子やクラス名，操作名などのリストを生成している．

次にポイントカットリストとジョインポイントリストとのマッチングを行う．該当したジョインポイントに対し，次の工程でアドバイスのモデルデータを挿入する．その際できるモデルデータを図3では新規モデルデータとしている．織り込むにあたっては，アスペクトの活性線から伸びたメッセージについてはsendIdやreceiveIdを織り込み先のオブジェクトIDに変更し，またIDの重複がないようチェックする．新規モデルデータをXMI形式に変換し，また描画情報であるPGML形式のファイルを出力する．その他IIOSSでモデルを扱うのに必要なファイルも出力する．

### 3.3 ステートチャート図の織り込み

ステートチャート図の織り込み処理について，同様に図3に沿って説明する．モデルデータの構築処理はシーケンス図の場合と同じである．ジョインポイントリストの生成については，2.3節で述べたジョインポイントの定義に従い，ジョインポイント間にあるイベントと遷移先のセットを抽出している．次にモデルのコンポジット状態をチェックし，アスペクトの検索を行い，ポイントカット記述とアドバイス処理本体のモデルデータを抽出する．ポイントカット記述について解析を行い，織り込み対象となるイベントと遷移先のセットをリスト化する．

それらをジョインポイントリストとマッチングし，該当したジョインポイントに対しアドバイスのモデルデータを挿入する．その際アスペクトのアドバイス処理本体におけるイベントの遷移元や遷移先のIDを変更する．その後のXMI形式，PGML形式等のファイル生成の処理はシーケンス図と同様である．

## 4 適用例

### 4.1 駆動方式の設計への適用例

シーケンス図を用いたアスペクト指向設計の例として，時間駆動やイベント駆動といった駆動方式の

アスペクト指向設計 [5] の例を紹介する．時間駆動とは時間周期に基づいて処理を行うことであり，イベント駆動とはイベントに応答して処理を行うことである．これらはリアルタイム性の要求に基づいて選択されるため，機能的要求に基づいて設計されるクラス構成とは独立に設計可能とする．

図 4 に，対象とする自動車制御の例の制御ブロック図を示す．制御ブロック図はデータフローのみを記述したものであるが，図 4 には駆動方式の説明のため制御フローを追記している．時間駆動に基づく場合は，図 4(a) に示すように，タイマによって周期的に処理が起動される．まずエンジントルクを計算し，次にスロットル開度を計算する．イベント駆動に基づく場合は，図 4(b) に示すように，入力イベントによって処理が起動される．

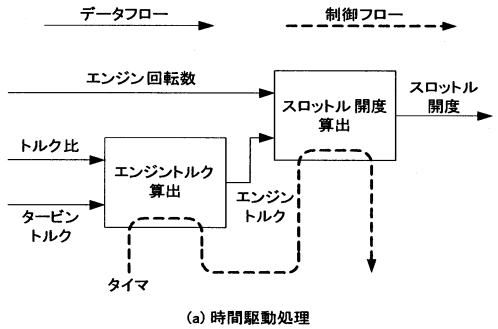


図 4 制御フローを記述した制御ブロック図

対象とした制御ブロック図に対応するクラス構成を図 5 に示す．EngineTorque はエンジントルクを表すクラス，ThrottleOpening はスロットル開度を表すクラスである．それらのクラスは，それぞれの値を表す属性 value を持つ．また，それらの値を計算し，更新する操作として update を，それらの返す操作として get を持つ．get 操作がデータフローに対応する．ThrottleControl クラスはスロットル制御全体を表すクラスであり，EngineTorque と ThrottleOpening を部分として持つ．その操作 exec は，時間駆動時に部分クラスの update 操作を呼び出すのに用いるが，駆動方式はアスペクトにより分離して記述するため空操作になっている．

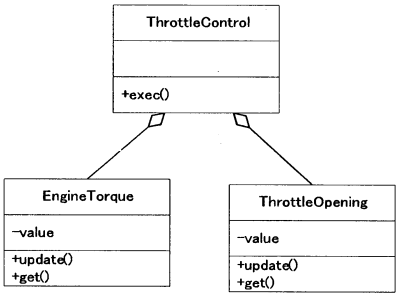


図 5 スロットル制御のクラス構成

この例の機能的側面は，制御ブロック図のデータフローにしたがって，エンジントルクやスロットル開度の計算を行うことである．これは，EngineTorque オブジェクトや ThrottleOpening オブジェクトの update 操作に対応する．

機能的側面である ThrottleOpening オブジェクトの update 操作に関するシーケンス図を図 6 に示す．ThrottleOpening オブジェクトの value を計算するには EngineTorque オブジェクトの value が必要であるため，ThrottleOpening オブジェクトの update 操作内で EngineTorque オブジェクトの get 操作を呼び出している．update 操作を呼び出しているオブジェクトの名前が書いていないのは，update 操作を呼び出すオブジェクトが場合によって変わるためである．EngineTorque オブジェクトの update 操作も，同様のシーケンス図によって記述される．

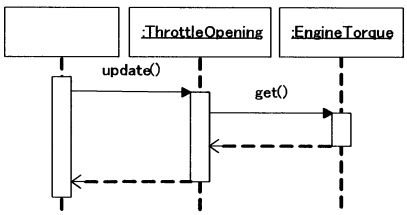


図 6 get 操作の呼び出し

上記機能的側面に対して，駆動方式を非機能側面として記述する．具体的にはEngineTorque オブジェクトと ThrottleOpening オブジェクトの update 操作の呼び出しシーケンスをアスペクトとして記述する．以下，時間駆動の場合とイベント駆動の場合のアスペクト記述について説明する．

(a) 時間駆動の場合

アスペクトとして時間駆動に基づく update 操作の呼び出しシーケンスを図 7 にシーケンス図で記述する．

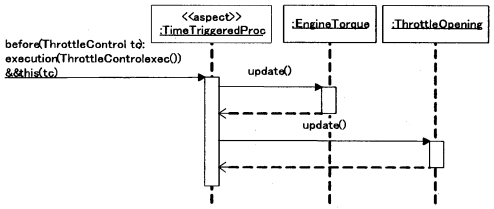


図 7 時間駆動処理アスペクト

時間駆動では時間駆動源に基づき，ThrottleControl オブジェクトが部分オブジェクトの update 操作をそれぞれ呼び出す．そこで，アスペクトとして TimeTriggeredProc を定義し，ポイントカットとして ThrottleControl オブジェクトの exec 操作の実行時を指定する．アドバイスの種類は before としているが，ThrottleControl オブジェクトの exec 操作は空であるため after でもよい．その選択位置に対し，EngineToruque オブジェクトと ThrottleOpening オブジェクトの update 操作を順次呼び出す処理を織り込む．また，this ポイントカットを用いているのは ThrottleControl オブジェクトが持つ EngineTorque オブジェクトと ThrottleOpening オブジェクトをアスペクト側から利用するためである．

次に時間駆動源による処理の起動を図 8 にシーケンスで記述する．時間駆動源としてはリアルタイム OS の周期タスクが挙げられる．

図 6 ～図 8 のシーケンス図のデータをモデルウィーバに入力すると，図 9 に示すシーケンス図を得る．時間駆動源によって ThrottleControl オブジェクトの exec 操作が呼び出され，その中に部分

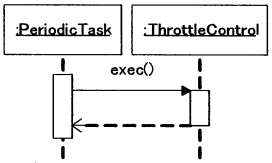


図 8 時間駆動源による処理の起動

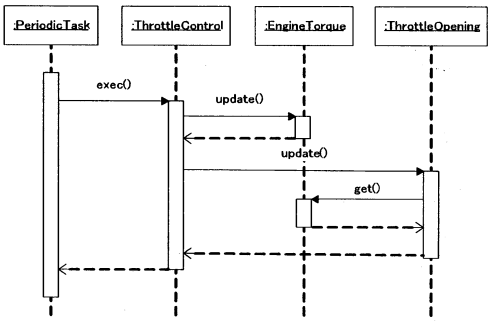


図 9 時間駆動処理の織り込み結果

クラスの update 操作の呼び出しが織り込まれていることがわかる．また，ThrottleOpening オブジェクトの update 操作内で Enginetorque オブジェクトの get 操作が呼び出されていることも確認できる．

(b) イベント駆動の場合

アスペクトとしてイベント駆動に基づく update 操作の呼び出しシーケンスを図 10 にシーケンス図で記述する．

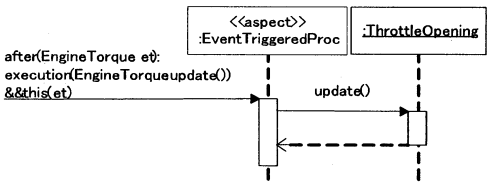


図 10 イベント駆動処理のアスペクト

イベント駆動では，イベント駆動源によって呼び出された EngineTorque オブジェクトの update 操作内で，データフローにおいて下流に位置する ThrottleOpening オブジェクトの update 操作を呼び出す．そこで，アスペクトとして EventTriggeredProc を定義し，ポイントカットとして EngineTorque オブジェクトの update 操作の実行時を指定する．下流オブジェクトの update 操作は，上流オブジェクトの update 操作が終わり次第

呼び出されるので，アドバイスの種類はafterとなる．その選択位置に対し，ThrottleOpeningオブジェクトのupdate操作を呼び出す処理を織り込む．

イベント駆動源による処理の起動を図11にシーケンス図で記述する．イベント駆動源としては外部入力やメッセージ受信の割り込みハンドラが挙げられる．ここではイベント駆動源をイベントハンドラとしている．

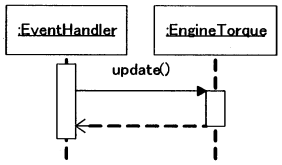


図11 イベント駆動源による処理の起動

図6・図10・図11のシーケンス図のデータをモデルウィーバに入力すると，図12に示すシーケンス図を得る．EngineTorqueオブジェクトのupdate操作が直接呼び出され，その処理が終わり次第ThrottleOpeningオブジェクトのupdate操作が呼び出されている．

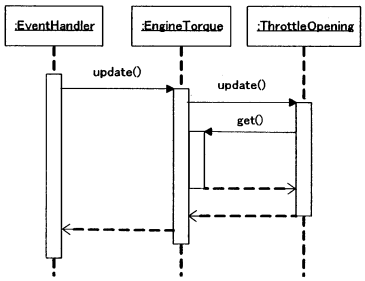


図12 イベント駆動処理の織り込み結果

以上のように，機能的側面であるデータフローから駆動方式に関する処理を分離して記述し，織り込むことができる．これにより，機能的側面として記述したクラスに対して，時間駆動を適用するかイベント駆動を適用するかをアスペクトにより独立に記述可能となる．

## 4.2 相互排除への適用例

ステートチャート図を用いたアスペクト指向設計の例として，相互排除を非機能的側面としてアスペクト記述する例を紹介する．具体的には，自動車のギアのシフトチェンジ時における相互排除をアスペクトで記述する．

まず機能的側面としてギアのシフトチェンジの状態遷移を表すステートチャート図を図13に示す．ここでは簡単のため状態を1速（1st）と2速（2nd）の2つのみとして説明する．1速の状態時にshiftUpイベントが発生した場合，gear属性の値を2に変更し，2nd状態に遷移する．2nd状態時にshiftDownイベントが発生した場合，gear属性の値を1に変更し，1st状態に遷移する．

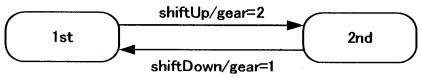


図13 ギアのシフトチェンジ

ここで，ギアのシフトチェンジの処理が複数のタスクで実行されるため，相互排除を行うことが必要であると想定する．アスペクトとして記述した相互排除の状態遷移を図14に示す．シフトチェンジを行う際は，gear属性が他のタスクからアクセスされている場合，それが解放されるまで待たなくてはならない．そこで，1st状態においてshiftUpイベントが起きた時と，2nd状態においてsiftDownイベントが起きた時に，gear属性へのアクセスを行う前にwait状態へ遷移する．またgear属性へのアクセスの再開をresumeイベントで表現する．

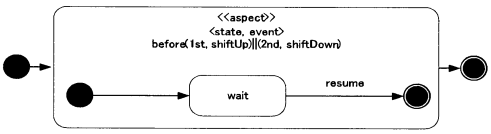


図14 相互排除のアスペクト

図13と図14をモデルウィーバに入力すると，図15に示すステートチャート図を得る．シフトチェンジを行う際，まず待ち状態に遷移し，処理の再開とともにgear属性の変更を行っている．

以上のように，相互排除の処理を機能的側面から分離してアスペクトとして記述し，織り込むことが可能になった．

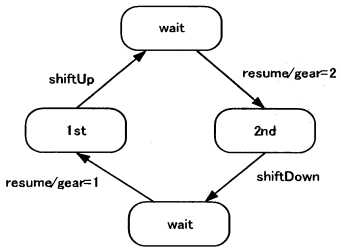

図 15 相互排除の織り込み結果

## 5 まとめ

本論文では，リアルタイムシステムを主な対象としたアスペクト指向設計環境として，シーケンス図とステートチャート図を対象に，UML の構文に独自の拡張を加えないアスペクト記述法を提案し，XMI ベースのモデルウィーバの開発を行った．これにより，モデルレベルで機能的側面と非機能的側面を分離して記述し，モデルウィーバによりそれらを織り込むとともに，同じ UML エディタで織り込み結果を確認することができる．本環境を用いることで，アスペクト指向に基づくインクリメンタルなモデル設計が可能になる．

今後の課題は，織り込みによって変化したモデル構造を他種のモデルにも反映させることでモデル同士の整合性を取れるようにすることである．

## 参考文献

[1] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. Loingtier, J. M. and Irwin, J., Aspect-Oriented Programming, Proc. of 11th European Conference on Object-Oriented Programming, pp. 220-242, 1997.

[2] Gal, A., Spinczyk, O. and Schröoder-Preikschat, W., On Aspect-Orientation in Distributed Real-time Dependable Systems, Proc. of 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, pp. 261-267, 2002.

[3] Mraidha, C., Gérard. S. and Benzakki, J., A Two-Aspect Approach for a Clearer Behavior Model, Proc. of 6th IEEE International Symposium on Object-Oriented Real-Time Distributed computing, pp. 213-220, 2003.

[4] de Niz, D. and Rajkumar, R., Time Weaver: A Software-Models Framework for Embedded Real-Time Systems, Proc. of 2003 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems, pp. 133-143, 2003.

[5] 横山孝典，組込み制御ソフトウェアのアスペクト指向に基づく開発法，情報処理学会論文誌，Vol. 47, No. 4, pp. 1185-1194, 2006

[6] Stein, D., Hanenberg, S. and Unland, R., A UML-Based Aspect-Oriented Design Notation for AspectJ, Proc. of 1st International Conference on Aspect-Oriented Software Development, pp. 106-112, 2002.

[7] Clarke, S. and Walker, R. J., Towards a Standard Design Language for AOSD, Proc. of 1st International Conference on Aspect-Oriented Software Development, pp. 113-119, 2002.

[8] Ho, W.-M., Jézéquel, J.-M., Pennaneac'h, F. and Plouzeau, N., A Toolkit for Weaving Aspect Oriented UML Designs, Proc. of 1st international conference on Aspect-oriented software development, pp. 99-105, 2002.

[9] Mraidha, C., Gérard, S., Terrier, F. and Benzakki, J., A Two-Aspect Approach for a Clearer Behavior Model, Proc. of Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 213-220, 2003.

[10] Domokos, P. and Majzik, I., Design and Analysis of Fault Tolerant Architectures Model Weaving, Proc. of Ninth IEEE International Symposium on High-Assurance Systems gineering, pp. 15-24, 2005.

[11] Kiczales, G., Hilsdale, E., Hugonin, J, Kersten, M., Palm, J. and Griswold, W. G., An Overview of AspectJ, Proc. of 15th European Conference on Object-Oriented Programming, pp. 327-353, 2001.

[12] 鈴木重徳，佐野元之，倉骨彰，垣花一成，IIOSS UML に基づく設計／開発環境のすべて，アスキー，2001.