サービス指向車載ソフトウェアの協調制御における タイミング設計方法の提案と評価

M2008MM023　永東　丈寛

指導教員:青山　幹雄

1. はじめに

現在，サービス指向アーキテクチャ(SOA: Service-Oriented Architecture)に基づく車載プラットフォームが研究されている[1]が，サービス間の協調のタイミングを厳密に保証する必要がある．本稿ではタイミング制約に基づきサービスインタフェースを拡張し，サービス協調のタイミング設計，検証方法を提案する．制約検証の支援ツール ContractValidator を実装し，有効性を評価した．

2. 車載ソフトウェアへの SOA の適用課題

サービス指向プラットフォームにおけるサービス協調のタイミング制約のモデルを図 1 に示す．自動車制御のためのサービス協調は，センサ入力を受けるサービス(Provided Service)，入力に応じた制御処理を実行するサービス(Control Service)，制御結果毎にアクチュエータへの出力を行うサービス(Required Service)によって行われる．そのため，センサ入力からアクチュエータ出力までの End-to-End のサービス協調の厳密なタイミング制約保証が必要である．

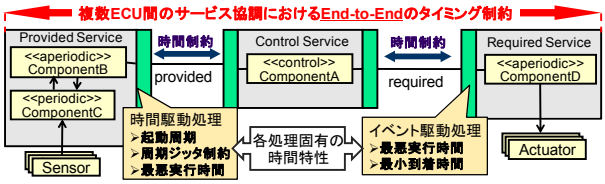

図 1:サービス協調のタイミング制約モデル

3. 関連研究

車載ソフトウェアの実行シナリオ毎の制約情報を定義可能なインタフェース(AI: Analytic Interface)を持つコンポーネントモデルに基づいて，WCET(Worst-Case Execution Time)を分析，評価する方法が提案されている[4]．しかし，サービス指向に基づいた制約分析は実現していない

また，QoS に基づく拡張インタフェース情報をモデル化するためのモデル記述言語として，QoSCL(QoS Constraint Language)が提案されている[3]．QoSCL は，QoS に関わる制約情報を記述するために UML2.0 のコンポーネントメタモデルを拡張したモデル記述言語である．しかし，サービスの特性に基づく制約記述形式は定義されていない．

4. 問題解決へのアプローチ

Design by Contract[2]に基づき，タイミング制約を形式的に記述できるサービスインタフェースの拡張を提案し，End-to-End のサービス協調の制約を分析可能な拡張インタフェースを定義する．さらに，モデル駆動開発に基づき，拡張インタフェースの制約の段階的な定義を可能とすることで，End-to-End のサービス協調のタイミング制約保証を実現する設計，検証方法を提案する(図 2).

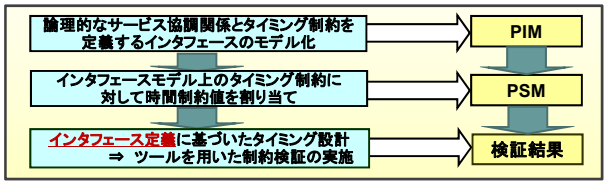

図 2:モデル駆動開発によるタイミング設計

5. モデル駆動タイミング設計方法

5.1. タイミング設計プロセス

提案するタイミング設計プロセスは，以下の 2 つのプロセスで構成される(図 3).

(I)　モデル化プロセス

A) 機能抽出

システムへの機能要求をユースケースとして記述し，ユースケース毎にシナリオを記述する．

B) サービスモデルの作成

制約定義を付加するサービスインタフェースとサービス協調の実行シナリオをモデル化する．

C) コンテキストの抽出

タイミング制約保証が必要な実行系列を，シーケンス図からコンテキストとして抽出する．

D) コントラクトモデルの作成

抽出したコンテキストとサービスモデルに基づき，各サービスの拡張インタフェースに制約を定義する．

(II)　検証プロセス

a) タイミング設計

サービスの振舞いから抽出される時間要求に基づいて決定するタイミング制約値を，コントラクトモデルの制約定義に割り当てる．

b) 制約検証

コントラクトモデルに割り当てた制約値に基づきコンテキスト毎の WCET を算出し，End-to-End のタイミング制約の検証を行う．

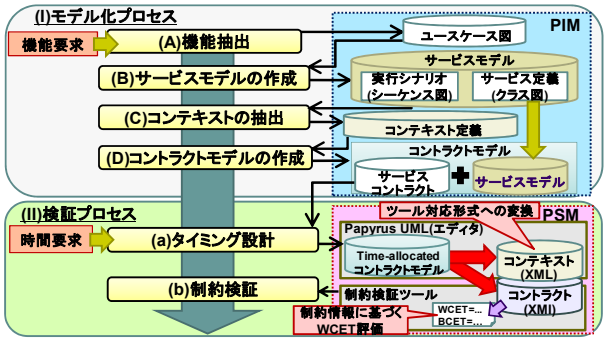

図 3: タイミング設計プロセス

5.2. モデル化プロセス

5.2.1. モデル化プロセスの工程

図 4 に示す以下のモデル化プロセスの工程を実行する．

(1)シナリオのモデル化

サービスモデルの構文定義とユースケース記述のシナリオに基づきサービス協調の実行シナリオをモデル化する.

(2)コンテキスト抽出

シーケンス図から End-to-End の実行系列としてコンテキストを抽出する.

(3)コントラクトのモデル化

抽出したコンテキスト毎の拡張インタフェース制約情報を,サービスの振舞いに基づき定義した時間特性で記述する.

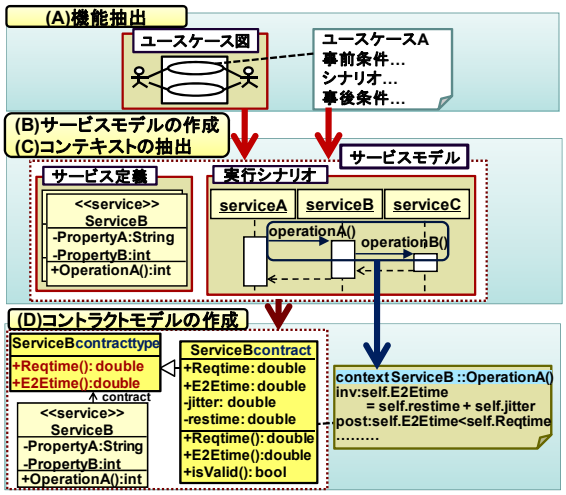

図 4: モデル化プロセスの実行工程

5.2.2. サービスモデル定義

サービスインタフェースの構文要素(サービス名, オペレーション, 属性)と, サービス協調の実行シナリオをサービスモデルとして定義する.

5.2.3. コンテキスト定義

外部イベントによるトリガ(センサ)を起点とするサービス協調の出力(アクチュエータ)までの実行系列をコンテキストとして定義する. コンテキストは, サービス協調の実行シナリオから End-to-End の実行系列として抽出する.

5.2.4. コントラクトモデル定義

抽出したコンテキスト中のサービスの拡張インタフェースをコントラクトとして定義する. コントラクトでは, コンテキスト中のサービスの振舞いから抽出される時間特性を用いて, OCL (Object Constraint Language)に基づく QoSCL の形式で End-to-End の制約を記述する.

5.3. 検証プロセス

検証プロセスでは, タイミング設計プロセスで制約値割り当てを行ったコントラクトモデルに基づき, ツールによる制約検証を行うことで, タイミング設計の妥当性確認を行う. 検証プロセスでは, 以下の 3 つのプロセスを実行する(図 5).

(1) 制約値の割り当て

時間要求に従って, コントラクトモデル上の制約記述に対してタイミング制約値を割り当てることで, 具体的なタイミング制約情報をコントラクトモデルに付加する.

(2) コントラクトモデル変換

モデルエディタ上で制約値を割り当てたコントラクトモデルを記述し, 制約検証ツールの入力形式である XMI へ変換する. また, コントラクトモデルの情報に基づきコンテキストを XML で定義する. モデル変換を行うツールとして, Papyrus UML[5]を利用した.

(3) タイミング設計の妥当性確認

変換した各モデル定義ファイルを制約検証ツールへ入力し, WCET に基づく制約検証を行う. 制約検証の結果から, タイミング設計の妥当性を確認する.

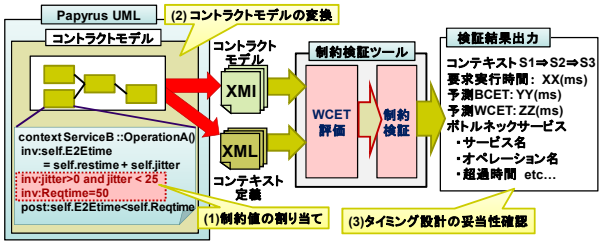

図 5: ツールを用いた制約検証プロセス

6. ContractValidator の実装

6.1. ContractValidator のアーキテクチャ

制約検証ツール ContractValidator を Eclipse 上で Java を用いて実装した. 図 6 にアーキテクチャを示す.

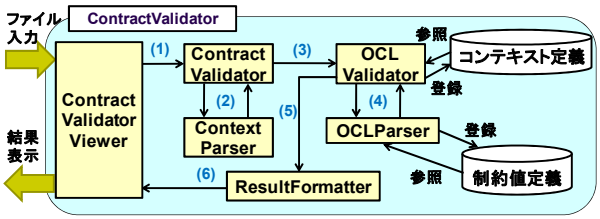

図 6: ContractValidator のアーキテクチャ

上述の ContractValidator のアーキテクチャに基づく実行の流れを以下に示す.

(1) ユーザ操作イベントに基づくデータ受け渡し

ContractValidatorViewer が提供する GUI 上で, ユーザが XMI 形式のコントラクトモデルと, XML 形式のコンテキスト定義のファイルを指定する.

次に, 検証開始のボタンをユーザが押下することでイベントリスナがイベントを検知し, 各ファイルのデータが ContractValidator へ受け渡される.

(2) コンテキスト分析

受け取ったコンテキスト定義ファイルデータの DOM から, コンテキスト定義要素を抽出するために, ContextParser へファイルデータを受け渡し, 抽出処理を行う.

(3) コントラクト及びコンテキスト定義のデータ受け渡し

抽出したコンテキスト定義要素とコントラクトモデルの DOM を OCLValidator へ受け渡す.

(4) 制約記述検証

コンテキスト定義要素に基づき, OCLParser に単一コンテキスト毎の制約検証を実行させる. コンテキスト定義に従って, コントラクトモデル DOM から検証対象コンテキストのサービスのコントラクトを抽出し, 制約定義に従った制約値の情報登録と制約情報に基づく WCET 計算を行う.

WCET に基づく制約検証を行い, コンテキスト毎のボトルネックや WCET, BCET 等の検証結果を抽出する.

(5) 検証結果受け渡し

全コンテキストの検証完了後, ResultFormatter へ検証結果データを受け渡す.

(6) 出力形式への変換

受け取った検証結果データを，GUI に出力可能な形式に変換する．そして，ContractValidatorViewer へデータを受け渡すことで，GUI 上に検証結果を出力する．

6.2. 実行環境と実装規模

表 1 と表 2 に実行環境と実装規模を示す．

表 1: 実行環境

OS	Windows Vista SP2
Java 実行環境	JRE1.6.0_07
Eclipse	3.5.0
XML パーサ	JAXP1.4

表 2: 実装規模

クラス数	8 個
LOC	1040 行

6.3. ContractValidator を用いた制約検証方法

ContractValidator は，指定されたコンテキスト定義(図 7b)に従った実行系列毎に，コントラクトモデル(図 7a)の XMI ファイルに記述される制約式を解析する検証を行い(図 7c)，タイミング制約値に基づく WCET を計算する．

検証後，コントラクトモデルで定義される制約記述を表示する(図 7d)．また，WCET に基づき各コンテキストの End-to-End の実行時間制約を満たさないサービスを抽出し，検証結果として表示する(図 7e)．検証結果によってボトルネックを可視化し，タイミング設計の妥当性確認を支援する．

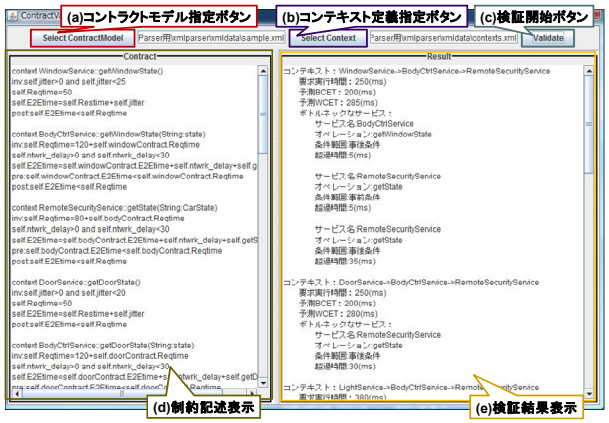

図 7: ContractValidator のユーザインタフェース

7. リモートセキュリティへの適用と評価

提案方法を自動車のリモートセキュリティへ適用し，有効性を評価した．

7.1. 適用対象システム

提案するタイミング設計方法を適用する対象として，自動車のリモートセキュリティを選択した．ドアロックの閉め忘れや，ハザードランプの消し忘れ等をした場合にユーザへの通知を行ううっかり防止機能を実現する．

7.2. サービスのモデル化とタイミング制約定義

リモートセキュリティを実現する一連のサービスとサービス協調のシナリオをモデル化し，制約保証が必要なコンテキストを抽出した(図 8)．窓，ドア，ライトの状態検知を起点としたうっかり通知と，ユーザの操作を起点としたリモート操作のコンテキストを抽出した．

また，抽出したコンテキスト毎に，サービスの振舞いに基づく時間特性を抽出し，制約定義を行うためのコントラクトモデルを作成した(図 9)．

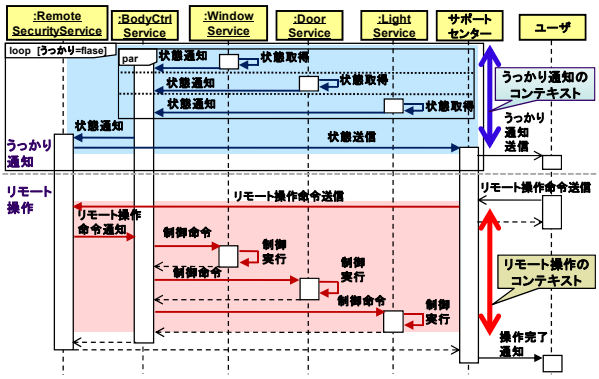

図 8: リモートセキュリティの実行シナリオとコンテキスト

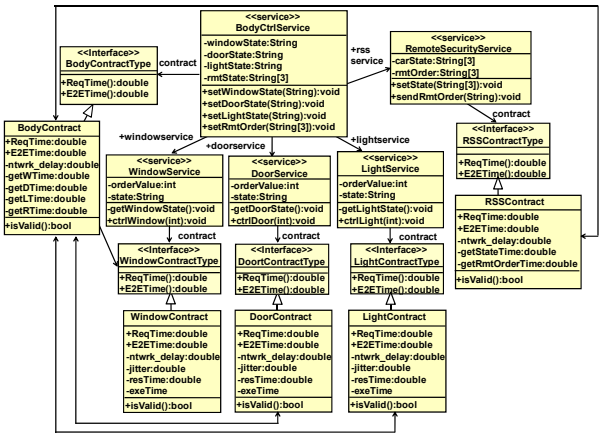

図 9: リモートセキュリティのコントラクトモデル

7.3. 制約検証とボトルネックの発見

コントラクトモデルで定義した時間特性を用いて，OCL に基づく制約記述を行い，各コンテキストの End-to-End の実行時間制約を定義する．図 10 にうっかり通知のコンテキストのタイミング制約記述を示す．コンテキストの時間要求に基づく実行時間制約を Reqtime，End-to-End の実行時間を E2Etime として定義し，各時間特性の制約値を割り当てた．

```
context WindowService::getWindowState()
  inv:self.jitter>0 and self.jitter<25
  inv:self.Reqtime=50
  inv:self.E2Etime=self.Restime+self.jitter
  post:self.E2Etime<self.Reqtime

context BodyCtrlService::setWindowState(String:state)
  inv:self.Reqtime=120+self.windowContract.Reqtime
  inv:self.ntwrk_delay>0 and self.ntwrk_delay<30
  inv:self.E2Etime=self.windowContract.E2Etime+self.ntwrk_delay+self.getWtime
  pre:self.windowContract.E2Etime<self.windowContract.Reqtime
  post:self.E2Etime<self.Reqtime

context RemoteSecurityService::setState(String:CarState)
  inv:self.Reqtime=80+self.bodyContract.Reqtime
  inv:self.ntwrk_delay>0 and self.ntwrk_delay<30
  inv:self.E2Etime=self.bodyContract.E2Etime+self.ntwrk_delay+self.getStatetime
  pre:self.bodyContract.E2Etime<self.bodyContract.Reqtime
  post:self.E2Etime<self.Reqtime
```

図 10: うっかり通知の制約記述

図 10 の例では，ネットワーク遅延 ntwrk_delay と周期ジッタ jitter が最大となる実行時間を WCET として算出した．

算出した WCET に基づく制約検証の結果，ボディ制御サービス，リモートセキュリティサービスの各 WCET が実行時間制約を超過することを発見した．

7.4. 提案方法の評価

(1) 制約記述能力の評価

コントラクトモデルの記述可能要素として，サービスの構文定義と制約定義がある．構文定義として，サービスのオペレーションと内部のプロパティが記述可能である．

また，制約定義の記述能力として，コントラクトモデルで定義した時間特性に基づく実行時間定義と実行時間制約定義が可能である．周期ジッタ等の可変な時間特性についても，形式的な記述に従って変化範囲を記述可能である．さらに，コントラクト間のロールを記述することで，コンテキストを定義するためのサービス間の依存関係を定義できる．

(2) 制約検証能力の評価

制約検証で対象としているシナリオは，センサ入力からアクチュエータ出力までの End-to-End のサービス協調の実行系列として定義されるコンテキストである．コンテキストは 1 つのセンサ入力から派生する実行パスの単一経路として考えられる．このような，ある入力に対する網羅的な実行系列に基づく制約検証によって，入力毎の End-to-End のタイミング制約を厳密に保証できる．

しかし，各実行系列が独立的に実行される場合はコンテキストに基づく検証が有効であるが，複数実行系列間の同期等，依存関係が存在する場合は，コンテキスト間の依存関係をモデル化した上での制約検証が必要となる．

8. 考察

(1) タイミング設計に対する考察

制約検証のシナリオにおいて，複数コンテキスト間で発生するタイミング競合を検証する必要がある．そのため，各コンテキストの起点となるセンサ入力のタイミングを分析し，同時並行して発生するコンテキストを特定することで，依存関係に基づく制約情報をモデル化する必要がある．

(2) 関連研究との比較に基づく考察

a) インタフェース定義要素の比較[4]

文献[4]で提案されるコンポーネントモデル上のインタフェースの制約定義は単一コンポーネントへの入出力に基づく実行シナリオを対象としている．本稿では，単一サービスではなく，複数サービス間の End-to-End の協調実行シナリオを対象とした制約定義を行うことで，End-to-End の実行時間制約検証を可能としている．

b) 制約記述言語のモデル記述能力の比較[3]

文献[3]の QoSCL では，サービスの特性に基づいた記述形式が定義されていない．本稿では，サービスの構文定義と End-to-End の制約定義に関わるモデル要素を拡張し，サービスの拡張インタフェース情報を定義している．

c) 車載ソフトウェア開発の抽象レベルでの位置づけ[6]

本稿で提案するコントラクトモデルを文献[6]の車載ソフトウェア開発抽象モデルと比較して考察する．図 11 に抽象モデルにおけるタイミング設計のモデル要素の位置づけを示す．文献[6]が定義する車載ソフトウェア開発の抽象レベルでは，システムの要求分析，設計までの機能的な視点で上位の 3 階層を定義している．本稿で提案するタイミング設計では，論理レベルのサービスの振舞いから時間特性を定義し，制約情報をモデル化しているため，これらの 3 階層へマッピングできるが，下位の実装レベルまで対応していない．

タイミング設計を実装レベルへ対応するためには，コントラクトモデルが定義する時間特性をハードウェアトポロジの視点から拡張し，実行時間制約の定義を厳密化する必要がある．今後は，定義すべき拡張情報の分析に基づき，コントラクトモデルを基盤とした設計レベルから実装レベルへの段階的なモデル化方法を考える必要がある．

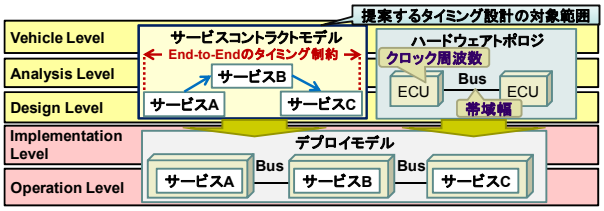

図 11: 車載ソフトウェア開発抽象モデルにおける位置づけ

9. 今後の課題

時間特性を抽出するための車載サービスの振舞いパターンを詳細化し，厳密な時間特性を定義する必要がある．また，本稿で定義したコンテキストは静的な End-to-End の実行系列であり，動的なサービスの振舞いにおける制約定義が困難である．そのため，動的なサービスの振舞いをモデル化するためのコンテキスト定義の拡張が必要である．

10. まとめ

拡張インタフェースモデルを用いたタイミング設計方法と制約検証方法を提案した．車載サービスの振舞いに基づく時間特性を定義し，時間特性を用いたタイミング制約を記述するコントラクトモデルを提案した．また，コントラクトモデルの制約情報に基づき，サービス協調のコンテキスト毎の WCET を算出し，WCET 検証を実現するツール ContractValidator を実装した．リモートセキュリティを例題として，提案方法の有効性を示した．

謝辞

本研究のご支援を頂いた，(株)デンソーの岩井明史氏と佐藤洋介氏に感謝する．

参考文献

[1] 青山幹雄，ほか，車載ソフトウェアのサービスプラットフォームのモデルとアーキテクチャ，自動車技術会，Oct. 2008, No.97-08, pp. 21-26.

[2] A. Beugnard, et al, Making Components Contract Aware, IEEE Computer, vol. 32, No. 7, Jul. 1999, pp. 38-45.

[3] S. Gerard, et al, Model Driven Engineering for Distributed Real-time Embedded Systems, Hermes Science, 2006.

[4] J. E. Kim, et al, Extracting, Specifying and Predicting Software System Properties in Component Based Real-Time Embedded Software Development, Proc. ICSE'09, May 2009, pp. 28-38.

[5] Papyrus UML, ver. 1.12, 2009, http://www.papyrusuml.org/.

[6] TIMMO (TIMing MOdel), TADL: Timing Augmented Description Language Version 2, 2009.