Vol.2018-ARC-230 No.28
Vol.2018-SLDM-183 No.28
Vol.2018-EMB-47 No.28
2018/3/8

AUTOSARカーネルにおける ターゲット依存記述のバージョン更新自動化手法

廣瀬 秀樹[1,a)]　高瀬 英希[1]　高木 一義[1]　高木 直史[1]

概要：車載ソフトウェアに含まれるターゲット依存のコンポーネントは，使用するターゲットごとに開発する必要がある．本研究では，AUTOSAR におけるカーネルに着目し，そのターゲット依存記述のバージョン更新を自動化する手法を提案する．バージョン更新がすでに完了しているターゲット向けのプロジェクトを解析することで，バージョン更新に伴って発生したソースコードへの変更を検出する．さらに，異なるターゲット間でプロジェクトを比較することで，ソースコード中のターゲット依存の記述を検出する．変更箇所に含まれるターゲット依存の記述を抽象化し，対応する情報は車載ソフトウェア開発者がドメイン固有言語を用いて記述する．これによって，カーネルのターゲット依存記述のバージョン更新が効率化できる．TOPPERS/ATK2 カーネルを対象とした適用事例によって，提案手法の有用性を示す．

1. はじめに

自動車の電子制御技術の向上により，車載システムにおけるソフトウェアの重要度がますます増している．車載ソフトウェアの高機能化に対する要求は高まっており，自動車に搭載される ECU（Electronic Control Unit）も増加傾向にある．近年の高級車では，搭載される ECU の個数は 140 以上にものぼる [1]．車載ソフトウェアの高機能化や ECU 数の増加に伴い，車載ソフトウェアの開発も大規模化および複雑化の一途を辿っている．

車載ソフトウェア開発の大規模化および複雑化を解決するため，複数の自動車関連企業から成るコンソーシアムによって AUTOSAR（AUTomotive Open System ARchitecture）仕様 [2] が策定されている [3]．AUTOSAR の目的は，車載ソフトウェアアーキテクチャの標準化，および，コンポーネント指向開発の実現による車載ソフトウェア開発の効率化である．AUTOSAR では，車載ソフトウェアを複数の層に分割し，それぞれの層内にソフトウェアコンポーネント（SW-C）が配置される構成となっている．アプリケーション機能を実現する上位層は，ECU ハードウェアに依存しない構造として規定されており，異なる車載システム間で再利用可能である．一方，ECU ハードウェアに依存する低位層に位置するコンポーネントは，使用する ECU ハードウェアごとに開発する必要がある．

ECU ハードウェアに依存するコンポーネントとして，本研究ではカーネルに着目する．カーネルは，一般的に，ターゲットに依存する部分（ターゲット依存部）およびターゲットに依存しない汎用的な部分（ターゲット非依存部）に分離して提供される．カーネルを更新する際には，更新の内容に応じて，ターゲット非依存部およびターゲット依存部のそれぞれを変更する必要が発生する．

TOPPERS/ATK2 カーネルについて，バージョン更新に伴うターゲット依存記述の変更を調査した．ターゲット依存部への変更は，新規ファイルの追加，既存ファイルの削除および既存ファイルへの修正の 3 種類に分類することができる．変更が発生したファイルのうち，既存ファイルへの修正が占める割合は約 80 %であった．さらに，既存ファイルへの修正は，コードの削除，コードの追加，行単位のコードの置換および行内におけるコードの置換の 4 種類に分類することができる．それぞれが占める割合は，コードの削除および追加の合計が 29 %，行単位のコードの置換が 24 %，行内におけるコードの置換が 47 %であった．

本研究では，車載ソフトウェア開発の効率化を目的として，カーネルのターゲット依存部を自動的に更新する手法を提案する．入力として，2 種類のターゲット向けのプロジェクトを与える．一方のターゲットについては，ターゲット依存記述がバージョン更新に追従できているとし，もう一方のターゲットをバージョン更新の対象とする．提案するツールによって，すでに更新が完了しているターゲット向けのターゲット依存部について，新バージョンおよび前バージョンのコードを比較し，変更点を検出する．さらに，両ターゲット向けの前バージョンのプロジェクト

[1] 京都大学
[a)] emb@lab3.kuis.kyoto-u.ac.jp

Vol.2018-ARC-230 No.28
Vol.2018-SLDM-183 No.28
Vol.2018-EMB-47 No.28
2018/3/8

target_serial.c

TARGET 1

```
#ifdef __f4002__
    while(count < VALUE){
        count++;
    }
    uart_send(data);
#endif

#ifdef __3580__
    data = receive(PORT);
    uart_send(data);
#endif
```

TARGET 2

```
#ifdef __f3550__
    if(count != VALUE){
        uart_send(data);
        set_counter();
    }
#endif
```

図 1 ターゲット依存コードの例

を比較し，ターゲットによって記述が異なる箇所（相違点）を検出する．ツールは，得られた変更点および相違点をもとに，ターゲット依存の変更を抽象化してフォームファイルおよび定義ファイルを出力する．車載ソフトウェア開発者は，中間生成物であるフォームファイルおよび定義ファイルを参照し，ターゲット依存情報をドメイン固有言語（DSL）で記述する．ツールにより，フォームファイルおよびターゲット依存情報を統合し，ターゲット依存部のコードを更新する．カーネルのバージョン更新自動化手法の適用事例として，TOPPERS/ATK2 カーネル [4] のバージョン更新を行う．

本研究の貢献は次の通りである．提案手法により，車載ソフトウェア開発者は，カーネルの開発にあたって，ターゲット依存の記述を簡易な DSL によって記述するだけで，使用するターゲットに対応したコンポーネントを生成することができる．ターゲット依存の変更点に対する記述を作成する際には，XML 形式で提供される定義ファイルを参照することで，適用すべき変更の内容を把握することができる．したがって，車載ソフトウェア開発者の負担が軽減され，車載ソフトウェア開発における生産性が向上することが期待される．

2. 準備

2.1 車載ソフトウェア開発における難点

車載ソフトウェアに用いられるカーネルは，使用するターゲットに依存する部分（ターゲット依存部）およびターゲットに依存しない汎用的な部分（ターゲット非依存部）に分けることができる．ターゲット依存部には，ターゲットごとに異なる値を定義しているコード，あるいは，構造自体が異なるコードが含まれる．

ターゲット依存部に含まれる，ターゲットごとに固有のコード（ターゲット依存コード）の例を**図 1** に示す．ターゲットごとに定義された値に応じて，`ifdef` 文によって分岐処理されている．さらに，処理の内容もそれぞれのターゲットによって固有である．

カーネルの機能を更新する際には，更新の内容に応じて，ターゲット非依存部およびターゲット依存部を変更する必要がある．ターゲット非依存部の変更はすべてのターゲットで共通である一方で，ターゲット依存部の変更はそれぞれのターゲットごとに行わなければならない．カーネル開発者は，変更が発生したコードが，ターゲット依存コードであるかどうかを把握する必要がある．さらに，ターゲット依存コードへの変更が発生した場合，使用するターゲットごとにコードの書き換えを行う必要がある．

2.2 関連研究

文献 [5] では，Complex Device Driver（CDD）を自動生成する手法を提案している．ドメイン固有言語によって CDD に含まれるターゲット依存記述を抽象化し，ターゲット依存の記述を分離している．車載ソフトウェア開発者は，ターゲット依存の情報のみを記述することによって，CDD を得ることができる．

文献 [6] では，プログラムに含まれるメソッドを対象に，バージョン更新を自動化する手法を提案している．似通った構造を持つ 2 つのメソッドのうち，一方を手作業で更新し，その更新をもう一方に自動的に適用する．構文解析および diff コマンドを組み合わせて使用することで，メソッドの編集を構文木への編集に変換している．文献 [7] では，コードクローン検出技術を使用して，デバイスドライバにおけるコードクローン（コード中に存在する類似したコード片）を解析している．さらに，コードクローンとして検出された部分をテンプレート，コードクローンでない部分をデバイスに固有情報として，デバイスごとにそれらを組み合わせてデバイスドライバを生成することを提案している．ただし，[6] および [7] は車載ソフトウェアを対象とした研究ではないため，ターゲット間における記述の差異を考慮していない．

3. カーネルのバージョン更新自動化手法

本研究では，カーネルのターゲット依存部のバージョン更新を自動的に実行する手法を提案する．カーネルがサポートする複数のターゲットのうち，少なくとも 1 種類についてターゲット依存記述がバージョン更新に追従できていることを前提とする．すなわち，バージョン更新が完了しているターゲットの更新前および更新後のプロジェクト，および，バージョン更新を行いたいターゲットの更新前のプロジェクトが利用可能であるとする．

提案手法の全体像を**図 2** に示す．差分解析部，コード生成部および統合部の 3 つのフェーズを持つ．各フェーズの詳細は 3.2 節で述べる．入力として，2 種類のターゲット向けのプロジェクトおよびターゲット依存情報を与える．既存ターゲットについては，前バージョンおよび新バージョンの両方のプロジェクトを入力する．開発対象ターゲットについては，前バージョンのプロジェクトを与える．まず，提案するツールが，バージョン更新に伴ってプロジェクトに適用された変更のうち，ターゲットによって内容が異な

Vol.2018-ARC-230 No.28
Vol.2018-SLDM-183 No.28
Vol.2018-EMB-47 No.28
2018/3/8

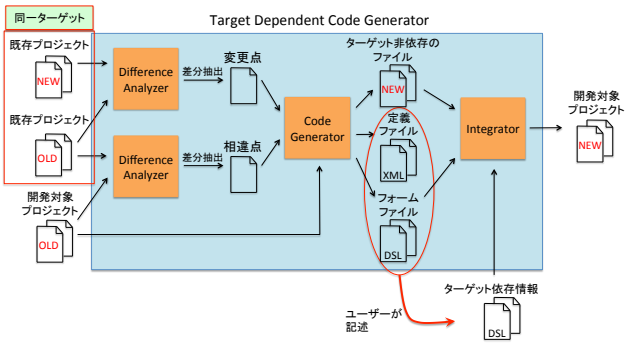

図 2 提案手法の全体像

target_serial.c

OLD

```
#ifdef __f4002__
    while(count < VALUE){
        count++;
    }
    uart_send(data);
#endif

#ifdef __3580__
    data = receive(PORT);
    uart_send(data);
#endif
```

NEW

```
#ifdef __f4002__
    while(count < TIME){
        count++;
    }
    data = process(data);
    uart_send(data);
#endif

#ifdef __3580__
    data = receive(PORT);
    data = process(data);
    uart_send(data);
#endif
```

図 3 ターゲット依存の変更の例

る変更を検出してその情報を車載ソフトウェア開発者に提示する．車載ソフトウェア開発者は，開発対象ターゲットに応じたターゲット依存の情報を DSL によって記述する．ツールは，ターゲット依存でない変更についてはそのまま更新し，ターゲット依存の変更については車載ソフトウェア開発者の記述したコードを参照して更新する．したがって，車載ソフトウェア開発者は，DSL によってターゲット依存のコードを記述するだけで，バージョン更新後のカーネルのターゲット依存部を得ることができる．

提案手法により，ターゲット依存の変更について，車載ソフトウェア開発者が DSL を用いて開発対象のターゲットに対応するコードを記述することで，開発対象のターゲット向けのカーネルのバージョン更新を自動的に実行することができる．ターゲット非依存の変更については，該当する箇所に自動的に変更を適用するため，車載ソフトウェア開発者はファイルを直接編集する必要がない．これにより，カーネルのバージョン更新を効率化することが可能であると考えられる．

3.1 DSL によるターゲット依存情報の抽象化

本研究では，変更が必要なターゲット依存の記述を抽象化し，ターゲット非依存の記述とターゲット依存の記述に分離することを提案する．提案手法では，変更が発生したファイルに含まれるターゲット依存記述に対する変更点を，DSL によって抽象化し，フォームファイルとして車載ソフトウェア開発者に提供する．さらに，変更点についての説明を定義ファイルとして提供する．車載ソフトウェア開発者は，フォームファイルにおいて抽象化された記述に対応するターゲット依存情報を，使用するターゲットに応じて記述する．これらのファイルを統合することで，使用するターゲットに応じて更新を適用する．

フォームファイルでは，変更が必要なターゲット依存の記述は，

`@<type>:<name>$`

のような抽象表現によって定義される．`<type>`は，発生した変更の種類を表す．Delete，Add，Replace および Rename の 4 種類があり，それぞれ，コードの削除，コードの挿入，行単位のコードの置換および行内の一部の文字列の置換を表す．`<name>`には，変更前のコードがそのまま記述される．

ターゲット依存情報は，抽象表現の名称および対応する具体的な記述を列挙することで記述される．フォームファイル中の抽象表現`@<type>:<name>$`に対応するコードを，

`@<name>:<codes>$`

のように，区切り記号:および終了記号$を使用して記述する．

定義ファイルは，抽象表現によって抽象化された変更箇所について，それぞれの変更内容に関する情報を提供する．定義ファイルは，XML 形式で提供される．1 つの`<exp>`タグが 1 つの表現に対応する．それぞれの抽象表現の名称，変更の種類，既存ターゲットにおける変更前の記述および変更後の記述が，それぞれ`<name>`，`<change-type>`，`<old>`および`<new>`によって整理して提供される．ターゲット依存の分岐構造が含まれる場合，その分岐に使用された定義が`<old>`および`<new>`にタグとして付加される．

車載ソフトウェア開発者は，フォームファイルおよび定義ファイルを参照して，ターゲット依存情報を記述する．例として，図 1 に示したターゲット 1 のコードに対して，**図 3** に示すような変更が発生した場合を挙げる．変更は 3 箇所存在する．そのうち 1 箇所は変数の名前を書き換える変更（Rename）であり，2 箇所は処理文の追加（Add）である．この場合における，ターゲット 2 のコードに対応するフォームファイルおよび定義ファイルの記述を**図 4** に示す．それぞれの変更点について，ターゲット 2 のコードの対応する箇所を抽象化している．定義ファイルにおいて，これらの抽象表現に関する情報が XML 形式で提供されている．

さらに，図 4 を提示された開発者が記述すべきターゲット依存情報の例を**図 5** に示す．フォームファイルに含まれる抽象表現のそれぞれについて，開発対象ターゲット向けのコードを DSL によって記述している．

Vol.2018-ARC-230 No.28
Vol.2018-SLDM-183 No.28
Vol.2018-EMB-47 No.28
2018/3/8

フォームファイル

```
#ifdef __3550__
    if(count != @Rename:VALUE$){
        @Add:data = process(data);$
        uart_send(data);
        set_counter();
    }
#endif
```

定義ファイル

```
<exp>
  <name>VALUE</name>
  <change-type>Rename</change-type>
  <old tag="__f4002__">VALUE</old>
  <new tag="__f4002__">TIME</new>
</exp>
<exp>
  <name>data = process(data);</name>
  <change-type>Add</change-type>
  <new tag="__f4002__">data = process(data);</new>
  <new tag="__f3580__">data = process(data);</new>
</exp>
```

図 4 フォームファイルおよび定義ファイルの例

```
@VALUE:TIME$
@data = process(data);:
data = process(data);$
```

図 5 ハードウェア依存情報の記述例

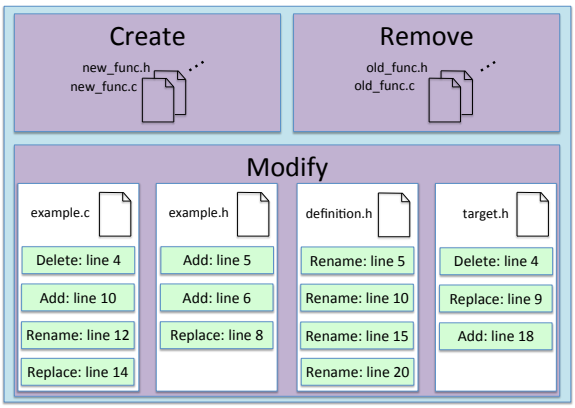

図 6 提案手法における変更点の分類

3.2 バージョン更新の自動化

提案するツールは，差分解析部，コード生成部および統合部の 3 つのフェーズを持つ．それぞれのフェーズについて，順に説明する．

3.2.1 差分解析部

差分解析部は，入力されたプロジェクト間の差分を解析し，変更点および相違点を抽出する．同一のターゲット向けの 2 つの異なるバージョンのプロジェクト間の差分を解析し，変更点として出力する．さらに，更新済みのターゲットおよび開発対象のターゲットについて，前バージョンのプロジェクトを解析し，相違点として出力する．変更点は，**図 6** のように分類される．

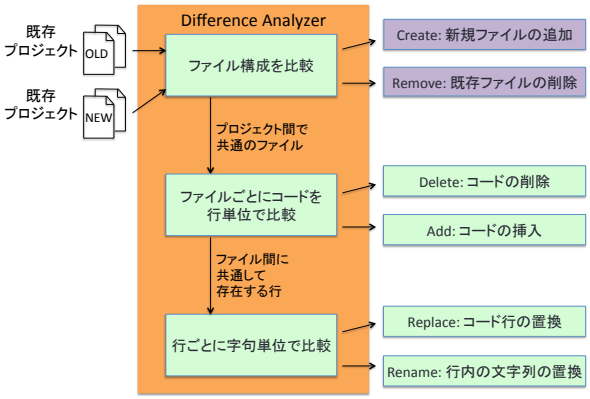

図 7 差分解析部の動作

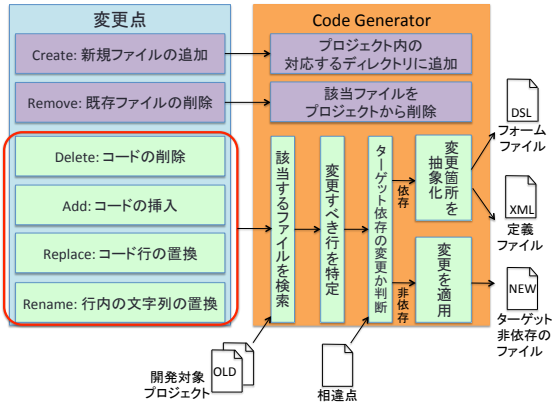

図 8 コード生成部の動作

ファイルごとに分類される変更点は，Create および Remove の 2 種類である．それぞれ，新規ファイルの追加および既存ファイルの削除を表す．

Modify に分類されたファイルにおける変更点は，さらに Delete，Add，Replace および Rename の 4 種類に分類される．それぞれ，コードの削除，コードの挿入，行単位のコードの置換および行内の一部の文字列の置換を表す．

差分解析部の動作を**図 7** に示す．差分解析部はまず，プロジェクトのファイル構成を比較する．新バージョンのプロジェクトにしか存在しないファイルがあれば，バージョン更新の際に追加されたものと考えることができるため，Create に分類する．一方，旧バージョンのプロジェクトにしか存在しないファイルは，バージョン更新に伴って削除されたと考えられるため，Remove に分類する．両方のバージョンに共通して存在するファイルは，Modify に分類し，そのソースコードを行単位で比較する．各行における変更を Delete，Add，Replace および Rename に分類する．

3.2.2 コード生成部

コード生成部は，バージョン更新に伴って発生する変更点およびターゲット間におけるプロジェクトの相違点をもとにターゲット依存の変更を抽象化し，フォームファイルおよび定義ファイルを生成する．ターゲット非依存の変更は開発対象プロジェクトに適用し，バージョンを更新する．

コード生成部の動作を**図 8** に示す．

まず，コード生成部は，プロジェクトへの新規ファイルの追加および既存ファイルの削除を行う．新しく追加されたファイルは，開発対象プロジェクト内の対応するディレクトリに追加する．既存ファイルの削除に関しては，プロジェクト内で一致するファイルを検索し，一致するファイルが見つかった場合にそのファイルをプロジェクトから削除する．

次に，ファイルごとに分類された変更点を処理する．まず，同一のファイルが開発対象プロジェクト内に存在するか検索する．一致するファイルが見つかった場合，そのファイル内で変更を適用すべき行を検索する．さらに，相違点をもとに変更箇所がターゲット依存コードであるかどうか判断する．相違点が存在する箇所への変更，および，ターゲット依存の分岐を含む記述の Add および Replace は，ターゲット依存の変更として扱う．ターゲット依存の変更の場合，変更箇所を抽象化してフォームファイルおよび定義ファイルを生成する．ターゲット依存の変更でない場合，変更をそのまま適用する．

3.2.3 統合部

統合部は，更新されたターゲット非依存のファイル，フォームファイルおよびターゲット依存情報を入力とする．車載ソフトウェア開発者が記述したターゲット依存情報を参照し，フォームファイルで抽象化されている変更箇所を補完する．更新済みのターゲット非依存のファイルと統合することで，開発対象のプロジェクトのバージョン更新を完了する．

4. 適用事例による有用性の評価

提案手法の適用事例として，TOPPERS プロジェクト [8] により提供されている TOPPERS/ATK2 カーネル [4] を使用して，カーネルのバージョン更新を行う．まず，提案手法を用いて，バージョン更新を自動的に実行するツール（バージョン更新ツール）を Python で実装した．実装の都合上，C 言語で記述されたファイルのみを対象としている．実装したバージョン更新ツールを用いて，**表 1** に示す 7 通りのバージョン更新を行った．

4.1 適用結果

提案手法の有用性を評価するため，まず，差分解析部が分類したファイルごとの変更点の分類の正確性を調査した．結果を**表 2** に示す．全事例において，既存ターゲットのプロジェクトから変更されたファイルを正しく抽出することができた．さらに，Modify に分類されたファイルについて，提案手法によって生成されたソースコードの正確性を調査した結果を**表 3** に示す．ターゲット依存の変更が検出され，フォームファイルおよび定義ファイルが生成されたファイルについては，正確に記述されたターゲット依存情報を統合部に入力し，更新後のファイルを調査対象とした．改行，スペース，コメントなどの動作に影響を与えない記述を除き，TOPPERS プロジェクトが提供しているソースコードと同一の記述が生成されているものを正とした．同一でない記述が含まれているファイルを誤とした．Modify が発生しているが，バージョン更新ツールが生成しなかったファイルを生成なしとした．全事例における 93 個の Modify が発生したファイルのうち，74 個を正しく生成することができた．正しく生成された 74 個のファイルのうち，フォームファイルおよび定義ファイルを参照してターゲット依存情報を記述したファイルは 3 個であった．

誤りのあった 4 個のファイルのうち，3 個のファイルは，バージョン更新の際に開発対象ターゲット向けのファイルのみに変更が発生したファイルであった．残りの 1 個のファイルは，バージョン更新の際に開発対象ターゲット向けのファイルのみにターゲット依存の分岐構造が追加されたファイルであった．これらの変更点は，既存プロジェクトから得られる変更点には含まれないため，該当箇所を変更することはできなかった．

4.2 考察および議論

まず，提案手法が使用できる前提条件について述べる．提案手法は，2 つの異なるバージョン間のプロジェクトを解析することによって，バージョン更新に伴う変更点を検出している．したがって，前提条件として，少なくとも一つのターゲットについて，カーネルのターゲット依存部の更新が既に完了していることが挙げられる．すなわち，一つのターゲット向けのカーネルの更新が完了している状態で，別のターゲット向けのカーネルのバージョン更新を行う必要がある局面において，提案手法は有用である．さらに，提案手法は，異なるターゲット間のプロジェクトのファイル構成を比較して，両方のプロジェクトに共通して存在するファイルのみを適用対象としている．したがって，ターゲット間でプロジェクトのファイル構成の類似度が高ければ高いほど，提案手法の有用性も高くなる．前述の適用事例においては，実装の都合上 C 言語で記述されたファイルのみを対象としたが，提案手法は言語にかかわらず適用することができる．

次に，提案手法では扱うことのできない変更について述べる．まず，提案手法は，Create および Remove に分類されるファイルにターゲット依存コードが含まれていた場合にその依存性を取り除くことはできない．次に，提案手法は，前バージョンでは異なるターゲット間で同一の記述であるが，新バージョンへの更新に伴ってターゲットごとに異なる記述をしなければならないファイルを正しく更新することができない．さらに，提案手法は，更新済みのターゲットでは変更が発生していないが，開発対象のターゲットでは変更が必要な箇所を更新することができない．

表 1 適用対象のプロジェクトおよびバージョン番号

番号	既存ターゲット	開発対象ターゲット	前バージョン	新バージョン
1	RH850F1H	HSBRH850F1L	1.4.0	1.4.2
2	HSBRH850F1L	RH850F1H	1.4.0	1.4.2
3	RCZBASEV850	FL850F1L	1.3.1	1.3.2
4	FL850F1L	HSBV850E2FG4	1.3.1	1.3.2
5	HSBV850E2FG4	RCZBASEV850	1.3.1	1.3.2
6	HSBV850E2FG4	RCZBASEV850	1.3.0	1.3.1
7	RCZBASEV850	HSBV850E2FG4	1.3.0	1.3.1

さらに，提案手法の改善の可能性について考察する．差分解析部は，1 種類のターゲットについて，2 つのバージョン間でのプロジェクトを比較することによって変更点を検出している．入力に 2 種類以上のターゲットを使用することにより，変更点の検出精度を上げることができると考えられる．同様に，相違点の検出精度も，より多くの種類のターゲットのプロジェクトを使用することで改善することができると考えられる．

最後に，提案手法の有用性について，一般的な開発方法と比較して議論する．提案手法を用いない一般的なカーネルのバージョン更新では，開発者は，バージョン更新において発生した変更点を整理し，それぞれの変更点がターゲット依存かどうか判断した上で該当する箇所を書き換える必要がある．一方，提案手法において，開発者が行う作業は，定義ファイルおよびフォームファイルを参照してターゲット依存情報を記述することのみである．さらに，ターゲット非依存の変更を開発対象のプロジェクトに適用する手間も省かれる．ゆえに，カーネルのバージョン更新を効率化することができると考えられる．

表 2 ツールが分類した C ファイルおよびヘッダファイル

番号	Create	Remove	Modify	合計
1	6	0	10	16
2	6	0	9	15
3	1	0	10	11
4	1	0	14	15
5	1	0	10	11
6	10	1	20	31
7	10	1	20	31
合計	35	2	93	130

表 3 ツールが生成した C ファイルおよびヘッダファイル

番号	必要数	正	誤	生成なし	成功率 [%]
1	9	6	0	3	67
2	10	6	0	4	60
3	14	9	0	5	64
4	10	9	0	1	90
5	10	10	0	0	100
6	20	17	2	1	85
7	20	17	2	1	85
合計	93	74	4	15	80

5. まとめ

本研究では，カーネルのターゲット依存部を自動的に更新する手法を提案した．適用事例として，TOPPERS/ATK2 カーネルに対して 7 パターンのバージョン更新を行った．修正が必要なファイルのうち，約 80 %について正しく更新を適用することができた．本手法により，ある 1 つのターゲット向けのカーネルの更新がすでに完了している局面において，別のターゲット向けのカーネルの更新が効率化されることが期待できる．今後の方針として，提案手法では検出しきれない変更点および相違点を検出する方法を検討することが挙げられる．

謝辞 本研究の一部は JSPS 科研費 16H02795 の助成による．

参考文献

[1] 櫻井 剛：自動車の組込みソフトウェアの現状: AUTOSAR および ISO 26262（組込みシステムの信頼性・安全性），日本信頼性学会誌， Vol. 36, No. 4, pp. 197–205 (2014).
[2] AUTOSAR: AUTomotive Open System ARchitecture, `http://www.autosar.org/`（アクセス日：2018 年 2 月 5 日）.
[3] Fennel, H., Bunzel, S., Heinecke, H., Bielefeld, J., Fürst, S., Schnelle, K.-P., Grote, W., Maldener, N., Weber, T., Wohlgemuth, F., Ruh, J., Lundh, L., Sandén, T., Heitkämper, P., Rimkus, R., Leflour, J., Gilberg, A., Virnich, U., Voget, S., Nishikawa, K., Kajio, K., Lange, K., Scharnhorst, T. and Kunkel, B.: Achievements and exploitation of the AUTOSAR development partnership, Technical report, SAE Technical Paper (2006).
[4] TOPPERS プロジェクト：TOPPERS/ATK2, `https://www.toppers.jp/atk2.html`（アクセス日：2018 年 2 月 5 日）.
[5] 廣瀬秀樹，高瀬英希，高木一義，高木直史：AUTOSAR における複合デバイスドライバのマイコン依存記述生成手法，情報処理学会研究報告，Vol. 2016, No. 4, pp. 1–8 (2016).
[6] Meng, N., Kim, M. and McKinley, K. S.: Sydit: creating and applying a program transformation from an example, *Proc. of ESEC/FSE'11*, pp. 440–443 (2011).
[7] Ma, Y.-S. and Woo, D.-K.: Applying a code clone detection method to domain analysis of device drivers, *Proc. of ASEC 2007*, pp. 254–261 (2007).
[8] TOPPERS プロジェクト：`https://www.toppers.jp/index.html`（アクセス日：2018 年 2 月 5 日）.