



## 2006 年度下期未踏ソフトウェア創造事業 採択案件評価書

### 1. 担当PM

高田 浩和(株式会社ルネサステクノロジ)

### 2. 採択者氏名

開発代表者:吉川 彰一 (フリー )  
共同開発者:なし

### 3. プロジェクト管理組織

株式会社メルコホールディングス

### 4. 委託金支払額

2,827,277 円

### 5. テーマ名

FlexRay のシミュレーション環境と共通 API の開発

### 6. 関連Webサイト

なし

### 7. テーマ概要

#### ・背景と目的

近年自動車のコストに占める割合の内、電子部品がますます増加傾向にあり、それに伴いソフトウェアの開発コードも増加傾向にあります。開発コストが増大する一方、

より低コストで品質のいいソフトウェアが求められており、その矛盾を解決するためにもソフトウェアの共通化や開発環境の整備は急務です。また、その電子部品をつなぐ車載 LAN で、現在の標準的な CAN は、データの通信方式や速度から、次世代車両に利用するには性能不足が懸念されています。

以上のことより、本テーマの目的は、次世代車載 LAN プロトコルとして注目されている FlexRay の開発環境の整備と FlexRay の普及に貢献することです。

#### ・開発内容

Windows や Linux 上で FlexRay のシミュレーション環境を提供することで、ソフトウェアの開発効率を向上させます。また、FlexRay を扱うための共通 API を定義して、シミュレーション環境で開発されたソースコードをターゲットプラットフォームでも流用可能にします。Windows や Linux である程度開発を進められることによるコスト削減と共通 API のライブラリ化によるソフトウェアの安定性が期待できます。

## 8. 採択理由

FlexRay は今後普及の見込まれる次世代車載 LAN プロトコルであり、そのソフトウェア開発環境整備についてのニーズは大きいと考える。開発者は、携帯電話アプリケーション開発の経験から、共通 API と PC 上で実行可能なシミュレーション環境が組み込みソフトウェアの効率的な開発を行う上で極めて大きなインパクトを持つとして、FlexRay のアプリケーション開発にもこのような手法を持ち込むことを提案している。FlexRay シミュレータを開発することで FlexRay アプリケーションの開発効率が格段に向上し、FlexRay 共通 API ライブラリを提供することでソフトウェアの共通化を図ることが可能になると期待できる。

実際の FlexRay アプリケーション開発にどの程度利用できるかは未知数であるが、新たなアプローチであり未踏性が高いことから、未踏プロジェクトとして採択することは適当と判断した。採択に際しては、開発内容と開発形態については申請時点からの変更があったことから、開発予算については見直し(減額)を行った。

## 9. 開発目標

本プロジェクトの目的は、組み込みソフトウェアの開発手法として、PC 上でのシミュ

レーション環境を用いた開発手法を提案することであり、具体的なテーマとして FlexRay のシミュレーション環境の開発をすることにより、FlexRay の開発環境の整備を行うこと、ひいては FlexRay の普及に何らかの貢献をすることを目指した。

本プロジェクトにおける開発目標は以下のとおり。

- FlexRay のシミュレーション環境の開発によって、FlexRay アプリケーションの基本的な動作を確認できるようにする。
- FlexRay の共通 API を策定し、ソフトウェアのモジュール化と再利用性を向上させる。共通 API のライブラリを開発し、FlexRay アプリケーション・プログラムの開発を容易にする。

## 10. 進捗概要

未踏プロジェクトの下期開発期間(2006年12月から2007年8月)において、前半に FlexRay 仕様の調査とシミュレーション環境の実装を、後半に共通 API のサポートと各 API のチェック、そしてデモ・アプリケーションの開発を行った。

途中、2ヶ月程度、開発者の会社業務が忙しく、未踏プロジェクトのための時間がとれなかったこともあり、当初の計画どおりに進まない部分もあったが、シミュレーション環境については概ね計画どおりに完成させることができた。8月前半の最終成果報告会の後も最後まで開発を継続したが、一部、同期関係の実装など、開発期間中には実装できなかった部分が残った。

## 11. 成果

本プロジェクトでは、次世代車載 LAN プロトコルとして注目されている FlexRay の開発環境の整備と FlexRay の普及に貢献、安定したソフトウェアの供給を目指し、AUTOSAR の定める API 仕様を元に、FlexRay アプリケーションの開発を可能とするシミュレーション環境の開発を行った。

開発成果の特徴は以下のようなものとなっている。

- PC 上でシミュレーション環境を構築することにより、実機なしに FlexRay アプリケーション・プログラムの開発・動作確認を可能としたこと。
- AUTOSAR の定める API 仕様を元に、FlexRay の開発を可能にしたこと。従来の実機を用いて開発する手法に比べ、開発のスピードアップ、多人数での共同開発、バグの縮小などが期待できる。開発環境としては、Windows と Linux に対応した。

以下に、開発成果の詳細を示す。

## 1. FlexRay シミュレーション環境の構築

FlexRay シミュレーション環境として、クライアント・サーバ方式のシミュレーション環境を構築した(図 1-1)。

今回のプロジェクトで開発したプログラムを以下に示す。

- サーバ
- クライアント用ライブラリ (Windows 版、Linux 版)
- クライアント・サンプルアプリ (Windows 版、Linux 版)
- クライアント・デモアプリ (Windows 版のみ)

具体的には、FlexRay の 1 ノードを 1 クライアントとして、TCP/IP 通信でサーバにつなぎ、サーバには複数のノード同士をつなげ、FlexRay のネットワーク構成の代わりとした。FlexRay の API を使用したソフトウェアから見ると、あたかも FlexRay のネットワークが構成されていて各ノードと通信が出来ているかのように振舞う。サーバでは、各クライアント同士のネットワークの構成を、様々なトポロジーを組み合わせながら設定することが可能となっている。また、各ノード、各接続部分のネットワーク通信のデータのモニタリングを行うことができるようになっており、サーバを介してネットワーク全体をデバッグ、モニタリングすることが可能となっている。

今回開発した FlexRay シミュレーション環境では、クライアント・サーバ方式を採用したことで、非常にすっきりとした構成とすることができた。FlexRay 開発環境としても、Java 言語およびフリーソフトウェアを使用して実装したことで、ホスト OS やプラットフォームを選ばない開発環境が実現できている。

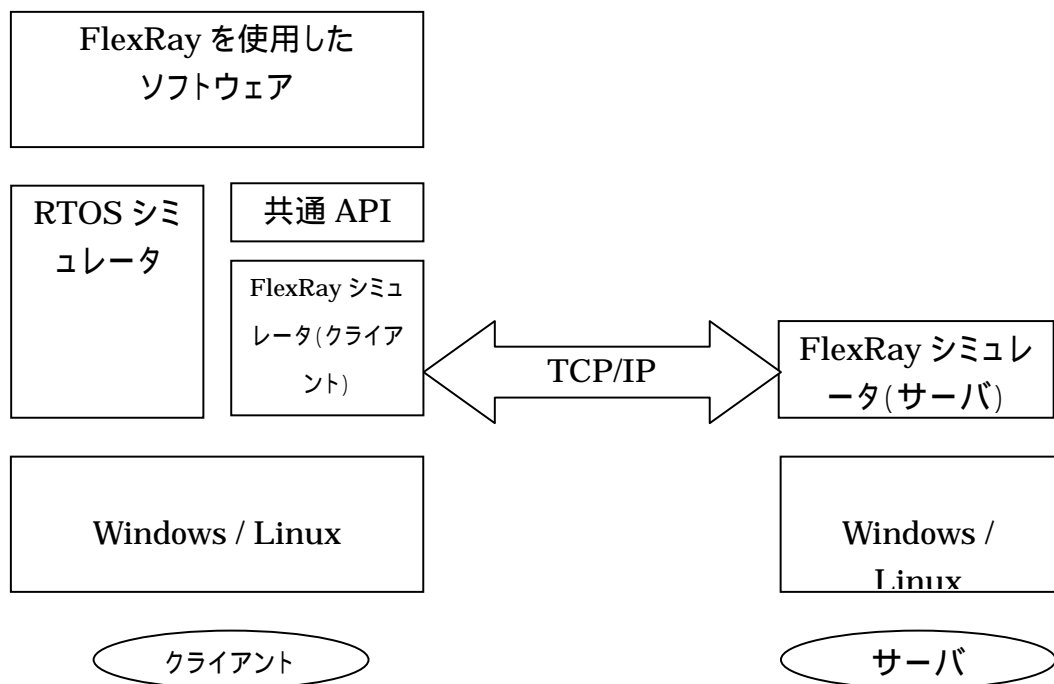


図 1-1 FlexRay シミュレーション環境

FlexRay のシミュレーション環境:

- RTOS シミュレータ TOPPERS/JSP Kernel 1.4.2
  - ◆ Windows シミュレーション環境 (Microsoft Visual Studio .Net)
  - ◆ Linux シミュレーション環境 (gdb)
- FlexRay シミュレータ (サーバ)
  - ◆ Java 2 Platform, Standard Edition, v 1.4.2 (Windows / Linux 共通)

FlexRay のシミュレーション環境を使用する開発環境:

- Microsoft Visual Studio .Net (Windows)
- gcc (Linux)

## 2 . FlexRay の共通 API

FlexRay の共通 API を定義し、シミュレーション環境上の開発でその API を使用して FlexRay のソフトウェアの開発を行う。それをそのままターゲット機へ移植することで、シミュレーションで得られたものと同等の効果をターゲット機上でも期待できる (図 1-2)。

共通 API としては、AUTOSAR が仕様策定・公開していた FlexRay 共通 API を採用した。本プロジェクトでは、AUTOSAR で定めた API のうち DRIVER API をサポートする。API 仕様の日本語版についても作成した。

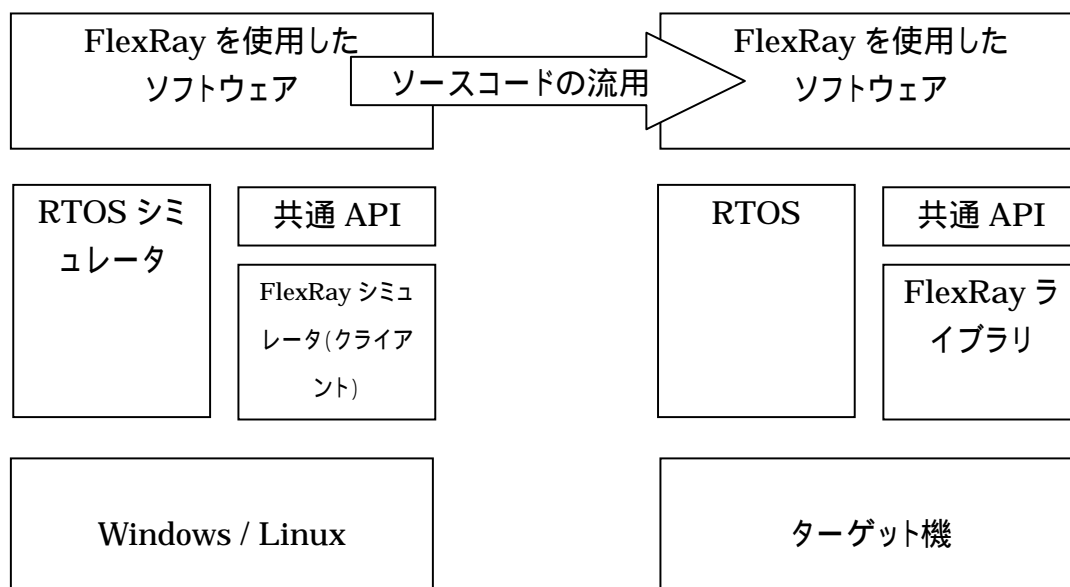


図 1-2 共通 API の利用

### 3. 使用例

FlexRay シミュレーション環境の使用例について、サーバの画面を図 1-3 に、サンプルアプリの実行画面を図 1-4 に示す。

Port	Power	POCState		Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10000/11000	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10001/11001	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10002/11002	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10003/11003	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10004/11004	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10005/11005	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10006/11006	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10007/11007	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10008/11008	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error
10009/11009	●	CONFIG	[REDACTED]	Tx(A)	Rx(A)	Tx(B)	Rx(B)	Access Error	Sync Error

図 1-3 サーバの画面

```

c:\ /cygdrive/c/eclipse3.2.2/workspace/flexray-emulator/client
[DEBUG] Emu_Receive: receive socket 2 byte data
[DEBUG] Emu_Receive: receive socket 4 byte data
[DEBUG] Emu_Receive: receive socket 1 byte data
[DEBUG] Emu_Receive: receive socket 1 byte data
[DEBUG] Emu_Receive: receive socket 1 byte data
[DEBUG] Emu_GetJob: called
[DEBUG] Emu_Receive: receive socket 1 byte data
[DEBUG] Emu_GetJob: returned = 1
[DEBUG] Fr_AckRelativeTimerIRQ: 1
[DEBUG] Emu_CallFunction: function = 0
[DEBUG] Emu_Send: send socket 62 byte data
[DEBUG] Emu_Receive: receive socket 2 byte data
[DEBUG] Emu_Receive: receive socket 4 byte data
[DEBUG] Emu_Receive: receive socket 4 byte data
[DEBUG] Emu_GetFunction: jp.gtco.flexray.server.autosar.FrAckRelativeTimerIRQ = 21
[DEBUG] Emu_CallFunction: function = 21
[DEBUG] Emu_Send: send socket 9 byte data
[DEBUG] Emu_Receive: receive socket 2 byte data
[DEBUG] Emu_Receive: receive socket 4 byte data
[DEBUG] Emu_Receive: receive socket 1 byte data
[DEBUG] Fr_TransmitTxLSdu: 2 1
[DEBUG] Emu_CallFunction: function = 0
[DEBUG] Emu_Send: send socket 57 byte data

```

図 1-4 サンプルアプリの実行画面

また、簡単なデモンストレーション用アプリケーション・プログラムとして、ハンドルによる車輪の操作を抽象化した通信プログラムの例を示す。ハンドルの制御タスクと、車輪の向きを制御するタスクが、FlexRay プロトコルにより通信を行うもので、図 1-5

に示すような GUI を使用してプログラムの動作が確認できる。それぞれのタスクがサーバを介して通信することにより、FlexRay プロトコルによる通信のシミュレーションを行うことができる。

このプログラムでは、キーで右に、キーで左にハンドルをきると、それに連動して車の車輪が動くことを見ることができるようになっている。その際の通信状況はサーバ画面上でモニタできる。また、サーバで強制的にエラーを起こすことで、通信が出来なくなること、その後、F11 ボタンで再起動すると、再びハンドルを操作できるようになること、などが視覚的に確認できる。

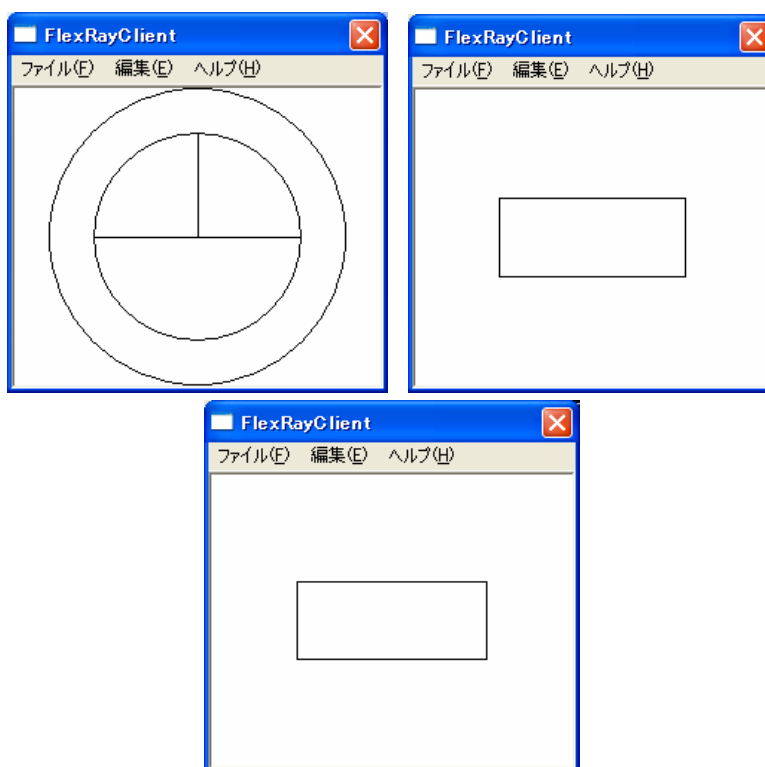


図 1-5 デモアプリの実行画面(ハンドルと2つの車輪)

## 12. プロジェクト評価

本プロジェクトでは、組み込みソフトウェアの開発手法として、PC 上でのシミュレーション環境を用いた開発手法を提案、その具体的なテーマとして FlexRay のシミュレーション環境と共通 API の開発を行った。

FlexRay の本格的な普及はまだこれからという状況であり、FlexRay に関して入手可能な情報も限られる中、未踏プロジェクトの開発期間中に当初の目標どおり FlexRay のシミュレーション環境を構築することができたことは素晴らしい。開発に際しては、

仕事との両立は非常に難しかったと思われるが、忙しい業務の合間をみつけて集中的に開発を行ったことは大いに評価できる。積極的に情報収集に務めたこと、最後まで諦めずに開発に取り組んだことなど、開発姿勢についても見習うべき点は多かった。

今後、組み込みソフトウェアの開発においても、シミュレーション環境を用いた開発手法が有効な手法の一つとなるものと思われる。ただし、実際に組み込みソフトウェアの開発に適用可能な仮想環境を実現するためには、IO デバイスも含めた組み込みシステム全体の動作がエミュレーションできることが重要となる。組み込みシステムの難しさは、多岐に渡る IO デバイスが接続されたシステムであり、その振る舞いはターゲットシステムのソフトウェアの挙動のみならず、外部の入出力の影響を多分に受けることにある。

その意味でも、ターゲットに接続される IO デバイスが汎用デバイスである場合、標準化された API を規定しその振る舞いをエミュレーションするというアプローチは、非常に有効である。デバイスモデルの API の抽象度に合わせて、ライブラリもしくはミドルウェアを整備することで、ソフトウェアの動作を保証することができれば、ソフトウェアの開発において仮想環境を採用することは十分可能と思われる。開発の方向性は間違っていないと思われるので、ぜひ今後もブラッシュアップしながら、実用化に向けて開発を継続して行って欲しい。

## 13. 今後の課題

未踏プロジェクトの開発期間を終え、FlexRay のシミュレーション環境の枠組みを構築し、その基本的な動作を確認することができた。開発した FlexRay シミュレーション環境については、PC 上で通信をシミュレーションし、AUTOSAR の API 仕様に基づくアプリケーション・プログラムの基本的な動作確認を行うことはできているが、アプリケーションの開発を容易にするには、上位の FlexRay ライブラリやミドルウェアといったものについても整備していく必要があると思われる。

今後の具体的な課題について、以下に列挙する。

- アプリケーション開発に有効な共通 API ライブラリおよびミドルウェアの整備。
- 具体的なターゲットデバイスに対するデバイス固有の使用への対応。

具体的なアプリケーションを想定した、シミュレーション環境のさらなるブラッシュアップとチューニング。時間精度の向上。