5C-06

組込みシステムにおけるGUI描画のための モジュール間通信の一開発手法

田中 宏平†

三菱電機株式会社

先端技術総合研究所†

1. はじめに

スマートフォンの普及により、ユーザがスマートフォンで体験する 60fps での GUI 表示が当たり前となり、他の様々な組込みシステムの GUI に対しても同等の描画性能が求められるようになりつつある。

一般的な組込みシステムの GUI は、GUI を描画する専用のモジュールが、システムの他のモジュール（ビジネスロジック）が管理するデータや接続されたハードウェアの状態を取得し、GUI を描画する。この際 GUI 開発の容易化のため、モジュール間通信を介して多数のビジネスロジックからデータを細切れに取得すると、通信のオーバヘッドが無視できなくなり、高速描画のニーズを満たすことが困難になる。

そこで本研究では、業務系システムなどの分散システムで用いられているインタフェース定義言語(IDL)を用いてモジュール間通信を定義し、定義からデータを蓄えるデータプール部を自動生成することで、開発の容易性を維持したまま、GUI の高速描画に対応する手法を提案する。

2. インタフェース定義言語

IDL（Interface Definition Language）は、ソフトウェアモジュールのインタフェースを記述するための記述言語で、プログラミング言語、通信手段などを抽象化して定義する。これまでにも数多くの IDL が提案されており、多くの IDL は専用の RPC（Remote Procedure Call）フレームワークのインタフェース定義のために用いられている。RPC フレームワークは、IDL を基にインタフェースとなるコードを生成する IDL コンパイラをもつ。IDL コンパイラが、同一の IDL から様々なプログラミング言語に対応したプログラムを生成することで、ソフトウェア流用率の向上や分散開発の容易化などの効果が生まれる。

A Development Method of Inter-Module Communication for GUI on Embedded System

†Kohei TANAKA, Mitsubishi Electric corporation, Advanced Technology R&D Center

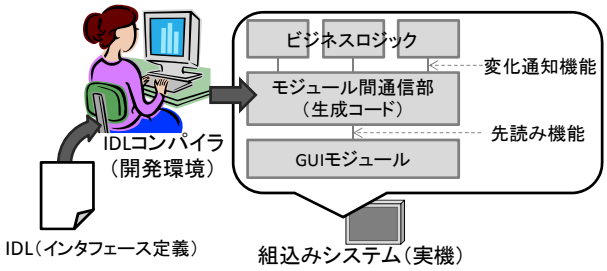

図 1: 提案手法のシステム構成

3. 提案手法

GUI モジュールのデータ取得の時間を削減するには、下記に示す 2 つの方式が考えられる。

方式A. データ取得の要求と応答を分離する。
方式B. データ取得を高速にする。

方式 A は、時間のかかるビジネスロジックからの応答を待たずに次の取得要求を発行可能にする（ノンブロッキングで要求する）ことで、GUI 内部処理で描画可能な表示オブジェクトのアニメーションやユーザ操作に対する高速な応答を実現する方式である。方式 B は、ビジネスロジックからの応答を先読みして GUI モジュールにデータの複製を保持しておくことで、データ取得時に通信を発生させない方式である。

方式 A はデータ取得時間を実質無視できるため高速な描画を実現可能であるが、応答のタイミングが予期できず GUI の状態数が増加し、品質確保が困難になるとの懸念から、本研究では、方式 B を実現する。

提案手法によるシステム構成を**図 1** に示す。提案手法では、モジュール間通信を定義した IDL から、データの先読みを行う機能やビジネスロジックでのデータ変化を GUI モジュールに通知する機能をコード生成することで、従来と同等の容易さで GUI に必要なデータ取得を実現する。以降で、順に本開発手法の流れを説明する。

まず、IDL コンパイラが IDL に定義されたインタフェースの中から取得されるデータを特定す

る。具体的には GetXXX などの関数定義から取得、IDL のプロパティ定義などから取得データを特定する。

次に IDL コンパイラは取得データを GUI モジュールが先読みするための機能とビジネスロジックからデータの変化を通知するための機能などのモジュール間通信部を生成する。データの先読みについては、GUI として表示する画面によって先読みするデータが異なるため、GUI 側からデータ単位で先読み又は先読み破棄を要求する関数である subscribeXXX()と unsubscribeXXX()を生成する。これらの要求はその場でデータを使用しないため、応答を後から受理するノンブロッキングな要求とする。変化通知については、ビジネスロジックにおいて GUI モジュールから先読みされたデータを更新した際に GUI に通知するイベントである XXXChanged()を生成する。他にデータを蓄積する蓄積部、蓄積部からデータを取得する関数なども合わせて生成する。

最後に IDL コンパイラから生成されたコードを用いてビジネスロジックと GUI を開発する。ビジネスロジックでは、従来データ要求に対してデータを返却する処理を実装していたものを、GUI からの先読み要求に対してのデータ返却、およびデータ変更時に変更を通知する実装に変更する。GUI は画面の中で必要になるデータを、前の画面構築時等で先読みする処理を追加で記述する。この追加の記述は GUI の構築に専用の設計ツールを使用している場合には、設計ツールの機能に組込めるため、開発時の実質の作業追加は発生しない。

続いて実機である組込みシステムの動作の流れを**図 2** に示す。

GUI モジュールがある画面を表示すると次の遷移先の画面で使用するデータを先読み要求しつつ、現在の画面に必要なデータを蓄積部から取得する。蓄積部が GUI モジュール内に存在するため、データ取得が高速になる。

ビジネスロジックの管理するデータがシステムの状況変化等で変化した場合は、GUI モジュールに対してイベントを発行する。この際データは蓄積部の値を更新するだけで、再描画するかは GUI 次第である。例えば定周期で描画する GUI の場合は、次の描画のタイミングで蓄積部の更新されたデータを参照して描画される。

4. 評価

GUI モジュールでの応答性向上の効果を確認するため、GUI モジュールからのデータ取得時間を計測した。計測箇所は、図 2 における 4-1 の呼出しから値が返却されるまでの時間である。評価環境を表 1 に、評価結果を表 2 左側に示す。結果から、約 20 倍の性能向上が期待できる。これは従来 60fps の画面描画中に約 100 回のデータ取得が可能だったものが、2000 回のデータ取得まで可能になることを示している。

次に開発の容易性を確認するため、コード実装量について従来との差を表 2 右側に示す。ビジネスロジック側で 1 つのデータに対して 4 行の追加で機能が実現可能である。なお表には記載していないが、GUI モジュールは先読み要求/破棄要求の 2 行追加され、大きな修正なく本提案手法が実現できることを示している。

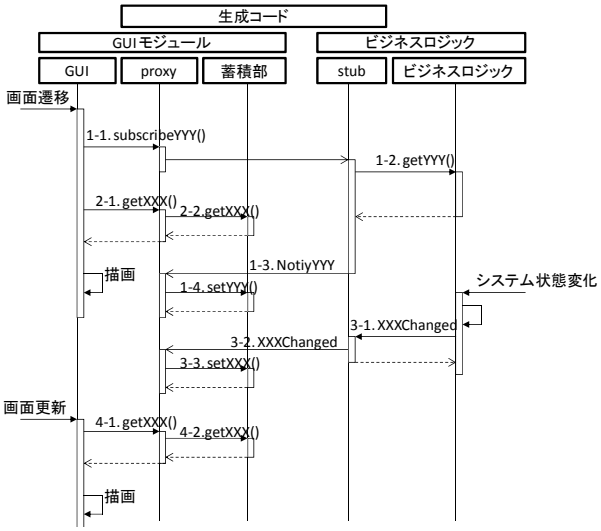

図 2: システムの動作の流れ

表 1: 評価環境

項目	仕様
評価環境	Core i7 870 2.94GHz, Ubuntu12.04
モジュール間通信	従来：C++間 Socket通信 提案：Javascript-C++ WebSocket通信
データサイズ	8 byte

表 2: 評価結果

	データ取得時間	ビジネスロジックコード量
従来手法	151.3 [us]	71行
提案手法	7.02 [us]	75行（更新通知分増加）

5. おわりに

本研究では、モジュール間通信で取得するデータをインタフェース定義から抽出し、先読み・管理する機能を生成することで、開発の容易性を維持したまま GUI の高速描画に対応する開発手法を提案した。提案手法を用いることで、従来から数行のコード修正でデータ取得時間を 20 分の 1 に短縮できる。

今後は本手法の適用で増加しているメモリ消費量を考慮した方式に拡張する予定である。