2B-05

組込みソフトウェア開発効率化のための インタフェース定義言語拡張の提案

田中 宏平†
三菱電機株式会社†
先端技術総合研究所

森田 知宏‡
三菱電機株式会社‡
先端技術総合研究所

1. はじめに

これまで組込み機器は，製品出荷時からソフトウェアを更新する必要がない，決定的な挙動が求められるといった理由から，利用するハードウェアに強く依存したプログラミングが多く行われてきた．また組込み機器の開発現場では，密なコミュニケーションでモジュール間のすり合わせを綿密に行うなどしていた．しかし近年，スマートフォンなどのソフトウェア規模の巨大な開発が増加し，従来方式での開発が困難になりつつある．そこで組込み機器向けのソフトウェアにおいても業務系システムの開発と同様に，機能間の結合を疎にしてモジュールを分割し，分散開発や流用開発を容易にするソフトウェアアーキテクチャの必要性が益々高くなってきている．

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）とその利用関係を示すモジュール図を用いることで，組込み機器のソフトウェア開発の分散・流用開発の容易化を支援する一方式を提案する．

2. インタフェース定義言語（IDL）

IDL（Interface Definition Language）は，ソフトウェアモジュール間のインタフェースを記述するための記述言語で，モジュール間のプログラミング言語，通信プロトコル，通信手段に依存しない抽象化された定義を記述するものである．IDL は開発プロセスの詳細設計前に作成する外部仕様書をソースコード化したものである．

IDL はこれまでにも数多く提案されているが，多くの IDL は，それ専用の 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 のコンパイラを持ち，IDL をコンパイルすることで，様々なプログラミング言語のインタフェースを生成する．これにより，同一の IDL から生成された機能を様々なプログラミング言語から利用できるようになり，ソフトウェアの流用率の向上などの効果が期待できる．

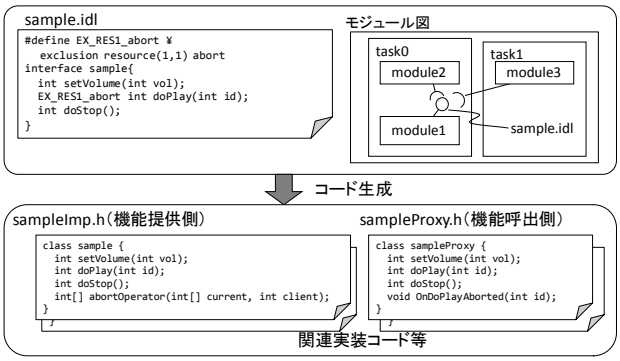

図 1：提案方式の概要

3. 提案方式

従来の IDL は基本的に業務系システム等のソフトウェアモジュール間の関係が疎な分散システムを前提としたものであり，組込み機器のような複数のモジュールで同一リソースを取り合うような密な関係なモジュールでの利用を考慮したものではない．そこで，組込み機器のソフトウェア開発で頻繁に問題となるリソースの競合を IDL とその利用関係を示すモジュール図を用いることで，開発の上流で解決する方式を提案する．これにより，組込みソフトウェアのモジュールの関係が疎になり，開発が効率化する効果が期待できる．

本研究で提案する IDL とモジュール図の概念を**図 1** に示す．提案方式では外部仕様である IDL とモジュール図から，機能呼出や機能提供を行う各モジュールの外部インタフェース部をコード生成する．以降で概要を順に説明する．

まず IDL 作成時にリソース競合が発生する可能

A Proposal of Interface Definition Language Extension to Reduce Embedded Software Development Efforts
† Kohei TANAKA, Mitsubishi Electric corporation, Advanced Technology R&D Center
‡ Chihiro MORITA, Mitsubishi Electric corporation, Advanced Technology R&D Center

性のあるインタフェースに対して IDL で修飾子 `exclusion resource abort` を設定する．`exclusion` はリソース競合を管理することを示す．`resource` ではモジュールからの要求処理時に使用するリソース量とその許容可能な量を定義する．例えば「音声再生処理機能を，モジュール X とモジュール Y から利用する」といったユースケースを考えた場合，1 つのデコード処理で 1 つのデコーダを専有するためリソース容量は 1，処理使用時のリソース量も 1 となる．デコーダを 2 つ同時に使用できる場合は，容量が 2 を規定する．`abort` は競合発生時の振る舞いを示したものである．競合発生時には動作できない処理を中断(`abort`)するか，保留(`suspend`)するかの 2 パターンの挙動が考えられるため，これを定義する．

次にこの IDL からコードを生成する．コード生成時には，IDL で定義されたインタフェースに加えて，リソース競合時の呼出元モジュールの優先度を判断する `abortOperator` と，処理が中断，または保留・再開されたことを通知する `OnDoPlayAborted` などを生成する．これらの関数は，生成されたコードから自発的にリソースの状態を判断して呼び出される．RPC の動作の概要と各機能の実装分担を**図 2** のシーケンス図に示す．機能提供側の実装者は `doPlay` などの提供機能の実装と `abortOperator` の処理を実装し，機能呼出し側の実装者は `doPlay` 等を呼び出すロジックと `OnDoPlayAborted` の処理を実装する必要がある．

4. 評価

提案方式と既存の IDL/RPC を機能面で比較した．比較には，Linux OS でよく使われている RPC である D-Bus[1]，Facebook 社が開発しオープンソースとして公開している RPC である Apache Thrift[2]，Google 社が開発しオープンソースで公開している IDL である Protocol Buffers[3]を用いた．本研究では提案方式を Apache Thrift の拡張により実装したため，通信方式や対応 OS，処理オーバヘッドなどの RPC に依存する部分は Apache Thrift と同じである．また，Protocol Buffers はデータ構造の定義による IDL であり，実通信部は持たないため，通信方式は「機能なし」とした．

結果を**表 1** に示す．競合調停機能は提案方式以外のいずれの IDL/RPC にも存在しない．競合調停機能が必要な場合には開発者が呼出側または機能提供側で実装する必要がある．通信方式は提案方式，D-Bus，Apache Thrift いずれも，1

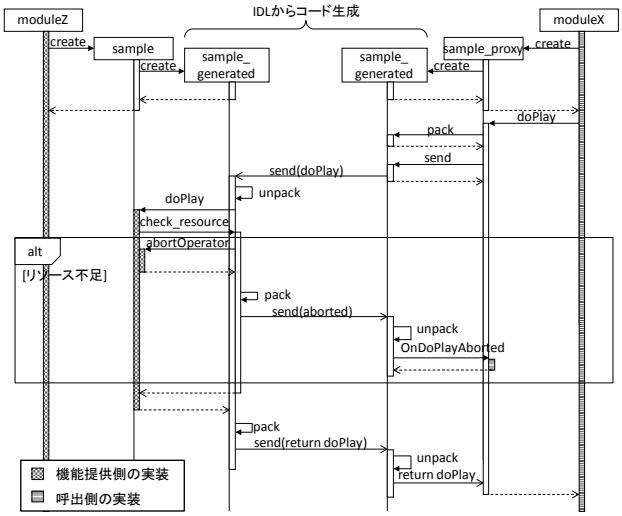

図 2: RPC 動作シーケンスと実装の分担

表 1: 既存 IDL との機能比較

	提案方式	D-Bus	Apache Thrift	Protocol Buffers
競合調停機能	○	×	×	×
通信方式	Socket他（1対多）	Socket他（1対多）	Socket他（1対多）	-
対応言語	1言語	少ない	多い	多い
通信データ量	小さい	通常	小さい	小さい

「-」：機能なし

つの機能提供部が複数のモジュールからの機能呼出しに対応する 1 対多の通信が可能である．対応言語は，提案方式では現状 C 言語向けの IDL コンパイラしか実装していないため 1 言語であるが，必要に応じて実装すれば多言語対応可能である．処理オーバヘッドは Apache Thrift や Protocol Buffers では，通信量削減のためデータシリアライズを工夫しており，通信オーバヘッドは比較的小さい．提案方式については，競合調停機能の追加でデータ量は増えないため，Apache Thrift と同等である．

5. おわりに

本研究では，IDL に対して組込みソフトウェア開発で必要となる排他制御を定義する仕組みを提案した．提案方式を用いることで，開発プロセスの上流でリソースの調停が設計されるため，下流の開発での過剰な連携を低減できる．

今後は提案方式をプロトタイプ開発などに適用し効果を確認する予定である．

参考文献

[1] D-Bus, http://www.freedesktop.org/wiki/Software/dbus/.
[2] Apache Thrift, https://thrift.apache.org/.
[3] Protocol Buffers, https://developers.google.com/protocol-buffers/.